A Variation on SVD Based Image Compression

Abhiram Ranade Srikanth S. M.* Satyen Kale†
Department of Computer Science and Engineering
Indian Institute of Technology
Powai, Mumbai 400076
ranade@cse.iitb.ac.in, srikanth_sm@symantec.com, satyen@cs.princeton.edu

Abstract

We present a variation to the well studied SVD based image compression technique. Our variation can be viewed as a preprocessing step *in which the input image is permuted as per a fixed, data independent permutation, after which it is fed to the standard SVD algorithm. Likewise, our decompression algorithm can be viewed as the standard SVD algorithm followed by a postprocessing step which applies the inverse permutation.*

On experimenting with standard images we show that our method performs substantially better than the standard method. Typically, for any given compression quality, our method needs about 30 % fewer singular values and vectors to be retained. We also present a bit allocation scheme and show that our method also performs better than the more familiar Discrete Cosine Transform (DCT).

We show that the original SVD algorithm as well as our variation, can be viewed as instances of the Karhunen-Loeve transform (KLT). In fact, we observe that there is a whole family of variations possible by choosing different parameter values while applying the KLT. We present heuristic arguments to show that our variation is likely to yield the best compression of all these. We also present experimental evidence which appears to justify our analysis.

1. Introduction

The use of Singular Value Decomposition (SVD) in image compression has been widely studied[1, 3, 9, 10]. If the image, when considered as a matrix, has low rank, or can be approximated sufficiently well by a matrix of low rank, then SVD can be used to find this approximation, and further this low rank approximation can be represented much more compactly than the original image. More specifically, suppose we are given an image A which we will assume for simplicity is an $N \times N$ real matrix. Then we first factor it into its SVD representation $A = U\Sigma V^T$, where Σ is a diagonal matrix with entries along the diagonal ordered in a non-increasing order, and U, V are orthogonal matrices [4]. Then a rank r approximation to A is the matrix $A_r = U_r\Sigma_r V_r^T$, where Σ_r is the top-left $r \times r$ submatrix of Σ, U_r consists of the first r columns of U, and V_r^T the first r rows of V^T. The SVD decomposition is interesting because U_r, Σ_r, V_r^T provide the best rank r approximation to A in the sense of packing the maximum energy from A. Furthermore, for compression, the decomposition is interesting because unlike A which has N^2 entries, the total number of entries in U_r, Σ_r, V_r^T are only $2Nr + r$. It often turns out that even with small r, the approximation A_r gets most of the energy of A, and is visually adequate. Hence the attractiveness of the method.

Our first contribution in this paper is to propose a slight modification to the above process, which gives us much better compression than the standard compression process described above. Our modification consists of a preprocessing step in which the entries in the image A are permuted with a data independent permutation S which we call shuffle (equation (1)). The resulting matrix $X = S(A)$ is then decomposed using SVD. Suppose $X_r = U'_r\Sigma'_r{V'_r}^T$ denotes the approximation obtained for X. We experimentally find that for the same value of r, the image $S^{-1}(X_r)$ is a much better approximation for A than A_r (Section 4). Thus for the same compression, we have better quality. We also provide a heuristic argument to justify our experimental finding (Section 3).

Our second contribution is to observe, that the original algorithm, which we will call $\mathcal{SVD}$[1], as well as our variation of it, which we will call $\mathcal{SSVD}$ (for shuffled SVD) can be both viewed as applications of the Karhunen-Loeve transform for compressing a single image (Section 2). The

*Currently at Symantec Corporation, Pune, India.
†Currently at Princeton University, Princeton, USA.

[1]To distinguish it from the mathematical operation of factoring a matrix, which we will denote as SVD.

standard definition of the Karhunen-Loeve transform (KLT) is concerned with compression of an ensemble of images rather than single image. However, by chopping up a single image into an ensemble of smaller images, we can use the KLT to compress a single image as well. We observe that the standard $\mathcal{SVD}$ and our new $\mathcal{SSVD}$ are nothing but different ways of partitioning the input image into an ensemble which can be compressed using the KLT. While it is well known that the KLT can be computed by a single SVD computation[2], we believe that the connection between the use of KLT to compress an ensemble of images and $\mathcal{SVD}$ (which compresses a single image) as well as our variation $\mathcal{SSVD}$ was not known.

Of course, to build industry standard compression programs one needs several techniques and tricks, of which SVD can only be one. For example, the familiar JPEG compression standard employs Discrete Cosine Transform (DCT) followed by sophisticated techniques for quantization, ideas such as zig-zag ordering, run-length coding, and Huffman coding. All these ideas contribute to the final compression. We expect that a similar array of ideas, for each level of the process, will have to be developed if $\mathcal{SSVD}$ is to compete with JPEG. Vector quantization is one idea that has been considered in the literature[3, 8, 9, 10]. In this paper we report on another strategy. We have developed a *bit allocation* scheme for representing the vectors constituting the matrices U and V (Section 5). We compare our scheme with a DCT coded with a quantization scheme described in [7].We find that our scheme works better than the DCT with quantization (although the full JPEG scheme, which employs many more tricks at different levels will work even better).

Most image compression algorithms in the literature typically divide the image into small blocks and then work on the blocks separately. While SVD based algorithms can be used with the entire image, typically they have been used on blocks[1, 3]. We compare $\mathcal{SVD}$ and $\mathcal{SSVD}$ at block level as well.

2. Comparison to KLT

Following Jain[5] we first define the KL transform for random vectors as follows. Let x denote a $p \times 1$ random vector, i.e. $x[i]$ are random variables over a certain probability space, for $0 \leq i < p$. For convenience we will index matrices and vectors starting from 0 rather than from 1. Let $R = E[xx^T]$ denote the autocorrelation matrix of x. Let R be diagonalized as $R = \Phi\Lambda\Phi^T$. Then the KL transform of x is $y = \Phi^T x$. Note that Φ depends on x, and hence both Φ and y are needed to represent x.

We next define[5] the KLT for an ensemble of images $A^0, \ldots, A^{p-1}$. For this we first linearly order the pixels in each image into a vector. Suppose x_i denotes the linearized version of A^i. Next we use the collection of the vectors as the probability space for the random vector x and adapt the previous definition. Let X denote the matrix formed by using x_i as its column vectors. The autocorrelation matrix for x is then simply $R = \frac{1}{p}XX^T$. Now let Φ be defined by the diagonalization $R = \Phi\Lambda\Phi^T$. Then the KLT of X is $Y = \Phi^T X$. Since Φ is data dependent, to represent X we need both Φ as well as Y.

Suppose $X = U\Sigma V^T$ is the SVD factorization of X. Then we have that $\Phi\Lambda\Phi^T = R = \frac{1}{p}XX^T = \frac{1}{p}U\Sigma^2U^T$, and hence $\Phi = U$ and $\Lambda = \frac{1}{p}\Sigma^2$. Further, $Y = \Phi^T X = U^T X = \Sigma V^T$. Thus, given Y we can obtain Σ and V simply by normalizing its rows. Thus, the KLT representation (i.e. Y and Φ) of the image ensemble represented by X is essentially the same as the SVD of X. This relationship is well known[2].

To use this to encode a single image A, we need to specify a way to break the image A into images $A^0, \ldots, A^{p-1}$ which are linearized and then made into the matrix X. Thus, we need to supply an operator P which can be used to construct $X = P(A)$. Clearly, the columns of $P(A)$ will be the images in the ensemble (after they are linearized). In general, P could be any reshaping/permutation operator, and we will see some examples next.

2.1. $\mathcal{SVD}$ as KLT

Let A be $N \times N$. Suppose we let P be the identity, i.e. $X = A$. The images in the ensemble are the columns of $X = A$. Thus the KLT of X will be exactly the $\mathcal{SVD}$ of A.

2.2. $\mathcal{SSVD}$ as KLT

Let A again be $N \times N$. Suppose further that $N = n^2$ with n integer.

We now choose P to be the shuffle operator S mentioned earlier. Informally, we form $X = S(A)$ by (i) breaking A into blocks of size $n \times n$, (ii) taking the ith block in row major order and arranging its pixels in row major order to produce the ith row of X. More precisely

$$X[\lfloor i/n \rfloor n + \lfloor j/n \rfloor, (i \bmod n)n + j \bmod n] = A[i, j] \quad (1)$$

To understand what the KLT of this $X = S(A)$ will give we need to understand how the images in this ensemble relate to A. Each image in the ensemble, i.e. a column of X is built by taking one element from each block of A, as defined above. Hence, we can think of each image in the ensemble as a low resolution sample of A.

2.3. Compression

As discussed earlier, to obtain compression we keep r columns of U, r rows of V, and $r \times r$ submatrix of Σ. The

size of the representation is then $r(d+1)$ where d is the sum of the dimensions of the matrix X. This representation will be good if X has low rank. This will happen if the columns of X are similar – equivalently if the images in the ensemble are highly correlated.

In the case of $\mathcal{SVD}$, the images in the ensemble are columns of the image, and clearly we expect adjacent columns to be similar. Notice however, that this similarity drops off as the distance between the columns increases. In the case of $\mathcal{SSVD}$, the images in the ensemble are low resolution subsamples of the original image. The jth point in every sample is taken from a single $n \times n$ block of the original image. Thus we expect these samples also to be correlated. In fact, the corresponding points on two samples are never more than a distance $2n$ apart in the original image. Whereas, in case of $\mathcal{SVD}$, the corresponding points on the columns can be as far apart as $N = n^2$. Thus we expect $\mathcal{SSVD}$ to perform better than $\mathcal{SVD}$.

3. Heuristic Analysis

Obviously we can partition the image in many other ways. For example, what if we break up A by rows? In this case, $X = A^T$. But SVD of A and A^T is essentially identical, except for reversing the roles of U and V. Another possibility is to break up A into blocks, rather than as low resolution samples. But notice that the rows of $S(A)$, as defined above are the blocks. Hence breaking up the image by blocks is equivalent for our purposes to breaking up by low resolution samples. Of course, there are many other possibilities. We could change the shape as well as the size of the blocks. Which of these is the best? We consider this question next in a heuristic manner.

Let our image A have M rows and N columns. Suppose we partition A into subimages by taking contiguous blocks of size $m \times n$. The matrix X constructed using this will have $\frac{MN}{mn}$ rows each of length mn. Clearly using $m = M$, $n = 1$ corresponds to the $\mathcal{SVD}$ scheme, while choosing $m = n = \sqrt{N}$ is the $\mathcal{SSVD}$ scheme assuming $M = N$. We will consider the performance of this family of partitioning schemes on a simple image.

Our image consists of a single straight line of constant intensity with length R and making an angle θ with the horizontal against a totally dark, zero valued background. The angle θ is chosen at random from the uniform distribution. This image is outrageously simple, but we believe it will indicate the general trends.

This line will pass through about $\frac{R\sin\theta}{m} + \frac{R\cos\theta}{n}$ blocks. Each block becomes a row of X. Thus, the rank of X cannot be larger than the above quantity. However, if the intersection of the line with the different blocks is identical, then the corresponding rows will be identical, and hence the rank would be lower. Unless the slope of the line has special relationships with m, n the intersections will be different; however when θ is close to 90° (alternatively, close to 0°) the line will be digitized into several long vertical (alternatively, horizontal) segments each of which could produce several identical intersections with different blocks. In this case the rank will just be $R\cos\theta$ (alternatively, just $R\sin\theta$). Thus we may conclude that the following quantity is a good estimate of the rank:

$$r = O\left(1 + \min\left(R\cos\theta, R\sin\theta, \frac{R\sin\theta}{m} + \frac{R\cos\theta}{n}\right)\right)$$

A simple integration shows that

$$E[r] = O\left(R\left(\frac{1}{m} + \frac{1}{n}\right)\right)$$

Above we have only argued an upper bound on the rank; however it will be seen that the same expression is a lower bound on the rank as well, to within a constant factor. For $\mathcal{SVD}$ ($m = M$, $n = 1$), the expected rank is thus $O(R)$, while for $\mathcal{SSVD}$ ($m = n = \sqrt{N}$), it is $O(R/\sqrt{N})$. Thus we should expect better compression with $\mathcal{SSVD}$.

We next consider the question: of all the different schemes for breaking up the original image into subimages, is there likely to be a scheme that works better than $\mathcal{SSVD}$? For this we will consider the size of the representation produced under the different schemes. As we have seen earlier, the size is $r(d+1)$. For our matrices X, we will have $d = \frac{MN}{mn} + mn$. Thus the expected size is:

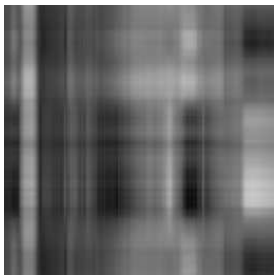
$$O\left(R\left(\frac{1}{m} + \frac{1}{n}\right)\left(\frac{MN}{mn} + mn\right)\right)$$

Clearly, for mn held constant this is minimized at $m = n$. With this choice the size becomes $O\left(\frac{R}{n}\left(\frac{MN}{n^2} + n^2\right)\right)$. This is minimized for $n = (MN)^{1/4}$. For the case $M = N$ we get $m = n = \sqrt{N}$, which is precisely the choice made in $\mathcal{SSVD}$.

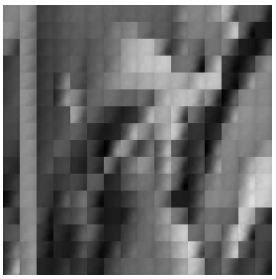
We note that a similar analysis will hold for images constructed by placing multiple lines *at randomly chosen inclinations*. Or in fact, if the image were to be constructed by placing, say ellipses or rectangles at random orientations. Of course, if the number of objects placed is very large, we will get to full rank in both $\mathcal{SVD}$ and $\mathcal{SSVD}$, but the point is that we will get to full rank with fewer objects in $\mathcal{SVD}$ than in $\mathcal{SSVD}$.

Of course, it is possible to construct images for which the standard $\mathcal{SVD}$ scheme will work better than $\mathcal{SSVD}$. For example, consider a dashed horizontal line. With $\mathcal{SVD}$, the matrix $X = A$ will still have rank 1; however with $\mathcal{SSVD}$ the matrix $X = S(A)$ will have larger rank, depending upon the gcd of n and the periodicity of the dashed line (i.e. the size of the dashes plus the blank separation). The question is whether we expect to have our images be dominated by dashed horizontal/vertical lines, or whether we

(a) Original Image



(b) $\mathcal{SVD}$ rank-2 approximation



(c) $\mathcal{SSVD}$ rank-2 approximation

(d) $\mathcal{SVD}$ rank-8 approximation

(e) $\mathcal{SSVD}$ rank-8 approximation

(f) $\mathcal{SVD}$ rank-14 approximation

(g) $\mathcal{SSVD}$ rank-14 approximation

Figure 1. $\mathcal{SVD}$ and $\mathcal{SSVD}$ when applied on image Lena

expect that lines at other inclinations will also be common. We expect the latter to be true, and hence $\mathcal{SSVD}$ likely to produce a more compact representation. It will be seen that this fairly naive analysis is supported by our experiments.

4. Experiments

We studied the performance of $\mathcal{SVD}$ and $\mathcal{SSVD}$ on a number of standard images such as Lena, Mandrill, Cameraman, etc., (images can be made available if needed).

We report the results on Lena in detail; while for the other images we only report whether and to what extent the results differed from those for Lena. The Lena image we used was of size 256×256.

4.1. Global SVD

By *global* SVD we mean application of $\mathcal{SVD}$, $\mathcal{SSVD}$ on entire image. Figure 2(a) compares $\mathcal{SVD}$ and $\mathcal{SSVD}$ when applied to the entire image. The x axis gives the representation size, i.e. $r(d+1) = 513r$. We see that at low PSNR, $\mathcal{SSVD}$ requires nearly half the size as $\mathcal{SVD}$. For higher, PSNR, the size for $\mathcal{SSVD}$ is still about 30% less than that for $\mathcal{SVD}$. Figure 1 gives an indicative visual comparison.

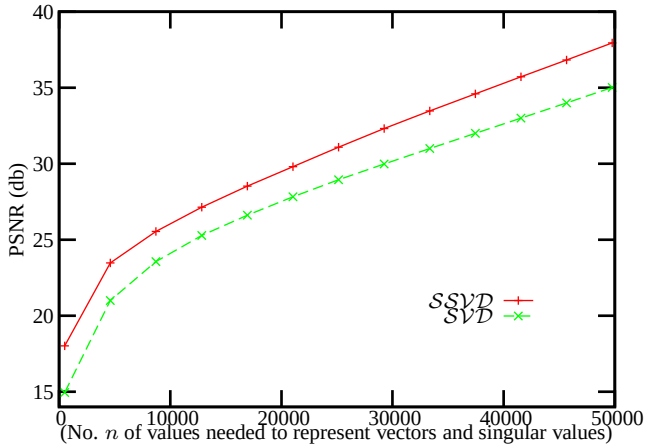
We also experimented with other block sizes[2]; 4×4, 4×8, 8×8, 8×16, 8×32, 4×64, 2×128. Most of these schemes fared worse than $\mathcal{SSVD}$ (which corresponds to block size of 16×16). Block sizes 8×8 and 8×16 gave performance generally comparable to $\mathcal{SSVD}$ (marginally better for low PSNR and worse for high PSNR). This generally validates the analysis for the best block size in section 3.

On most images global application of $\mathcal{SSVD}$ gives performance superior to $\mathcal{SVD}$. Similar to the performance on image Lena, $\mathcal{SSVD}$ fares significantly better than $\mathcal{SVD}$ on most images like Peppers, Barbara, Cameraman etc. On a very few images like Mandrill, the performances were close, while $\mathcal{SSVD}$ still yielded better performance. The worst case we encountered was image Goldhill where $\mathcal{SVD}$ very marginally dominates $\mathcal{SSVD}$.

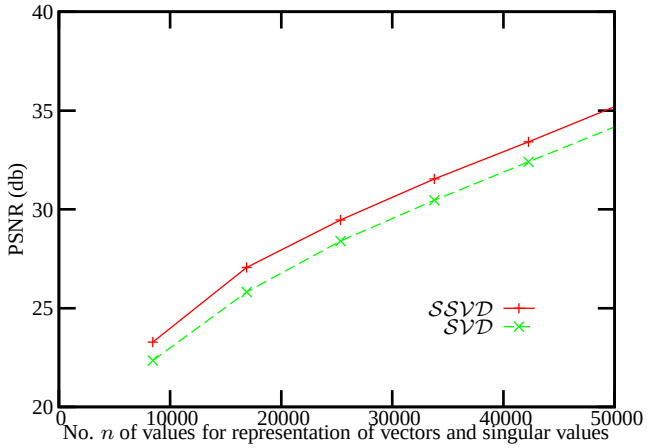
4.2. SVD on Blocks

In most studies of SVD based compression, SVD was applied not to the entire image but to individual blocks. Such *local* application is probably more suitable for the subsequent Vector Quantization (VQ) step used by many

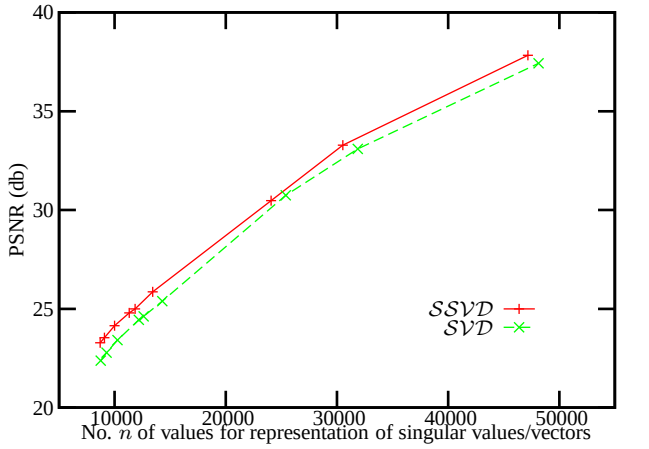
[2]The size of the representation in this case is $r(\frac{MN}{mn} + mn + 1)$ as noted earlier.



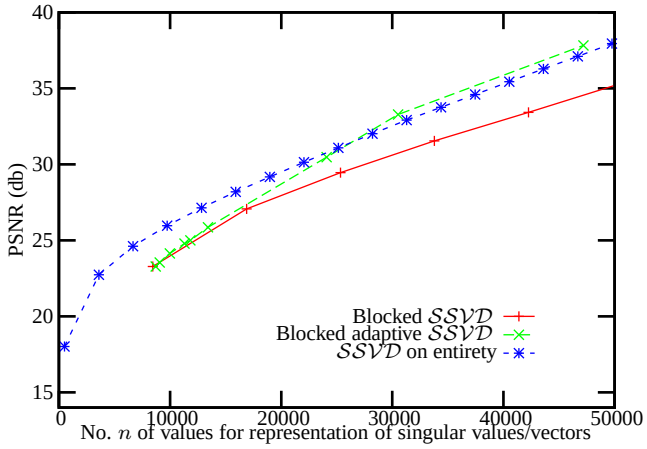
(a) $\mathcal{SVD}$ and $\mathcal{SSVD}$ when applied on entire image



(b) $\mathcal{SVD}$ and $\mathcal{SSVD}$ when applied on 16×16 blocks of the image



(c) 16×16 blocks, rank chosen adaptively.



(d) $\mathcal{SSVD}$ Variations

Figure 2. Various ways of applying $\mathcal{SVD}$ and $\mathcal{SSVD}$ to an image

researchers[3, 10]. So we decided to compare $\mathcal{SVD}$ and $\mathcal{SSVD}$ in this setting too.

Figure 2(b) shows the comparison when $\mathcal{SVD}$ and $\mathcal{SSVD}$ are applied on the 16×16 blocks of the image individually. In this experiment we used the same rank r for all blocks. As before we plot PSNR for the entire image against the size of the representation for all blocks together. We see in this case that, $\mathcal{SSVD}$ outperforms $\mathcal{SVD}$. We however, note that the performances are close.

Figure 2(c) shows the comparison when we represented each block to bound the fractional energy error ϵ to a fixed value. Specifically, suppose that σ_i's are the singular values of a given block. For this block we chose r such that $\frac{\sigma_1^2+\sigma_2^2+\cdots+\sigma_r^2}{\sum_j \sigma_j^2} \geq 1-\epsilon$. Different points of the curves were obtained by selecting different values for ϵ. Even in this case, we see that $\mathcal{SSVD}$ performs better than $\mathcal{SVD}$ and that the performances are close.

Although $\mathcal{SSVD}$ is seen to perform better in this case, we find from our experiments on a larger set of images (like Mandrill, Peppers, Cameraman) that the performances of $\mathcal{SVD}$ and $\mathcal{SSVD}$ are indistinguishable, when applied locally on the individual blocks of the image.

4.3. $\mathcal{SSVD}$ variations compared

Figure 2(d) shows in a single view the performance of the different variations of $\mathcal{SSVD}$ we have used: global (applied to entire image), local (applied to blocks), local with adaptive choice of ranks for each block. It appears that the non-adaptive local performs the worst (but still better than its $\mathcal{SVD}$ counterpart), while the local adaptive variation is

comparable to the global.

Our preliminary conclusion is that global $\mathcal{SSVD}$ is among the best (if not the best) of all the variations, and hence we have chosen it for the analysis/experiments of the next section.

5. A Bit Allocation Strategy

Since the first few singular values/vectors pack most of the image energy, it makes sense to allocate more bits to them. We now give a scheme for this.

Let $X = U\Sigma V^T$. Then we may also write $X = \sum_{k=1}^{r} \sigma_k u_k v_k$ where $r = \text{rank}(X)$, σ_k is the kth diagonal entry of Σ, and u_k and v_k are the column vectors of U, V respectively[6]. Alternatively, $x_{ij} = \sum_{k=1}^{r} \sigma_k u_{ki} v_{kj}$.

We will represent each element of u_k and v_k to w_k bits, with uniform quantization. Then the error in each element will be bounded by $Q_k = \frac{1}{2^{w_k}}$. Thus the error in x_{ij} is about $\sum_k \sigma_k Q_k (u_{ki} + v_{kj}) \leq \sum_k 2\sigma_k Q_k$. Thus the mean square error (MSE) of the image is $\frac{1}{mn} \sum_{ij} \left(\sum_k 2\sigma_k Q_k\right)^2$. Thus this must be minimized subject to the natural constraint on the total size of the representation i.e. $\sum_k w_k = w$. Using Lagrange's method of multipliers we obtain

$$w_k - w_{k+1} = \lg(\sigma_k) - \lg(\sigma_{k+1}),\ 1 \leq k \leq r \quad (2)$$

Now all w_k can be determined.

5.1. The Full Scheme

The values of σ_k could also be represented with varying precision; however these are much fewer than the u_{ki} and v_{kj} entries. So we represent σ_k as 32 bit floating point numbers.

The vectors u_k, v_k are stored as per the scheme described above, except for one additional idea. For each vector u_k, we store two additional values $u_{k\max}$ and $u_{k\min}$ which denote the maximum and minimum of the values in u_k. Then the w_k bits allocated to store each vector are used to represent each value within the range $[u_{k\min}, u_{k\max}]$, rather than the range [-1,1] which in general will be considerably larger. We store the maximum and minimum only in 8 bits each. The vectors v_k are also represented similarly, of course. We have found that storing the maximum and the minimum for each vector improves the compression. Clearly, more variations of this kind may be possible, and perhaps greater exploration is needed.

Figure 3 shows the performance of $\mathcal{SSVD}$ using the full bit allocation scheme. The performance of a DCT scheme is also shown. For applying DCT we divided the image (Lena) into blocks of size 8×8. The DCT coefficients were quantized according to quantization matrix specified by [7]. The six bit-allocation schemes for DCT coefficients given by [10] were used to obtain the six points on DCT curve shown.

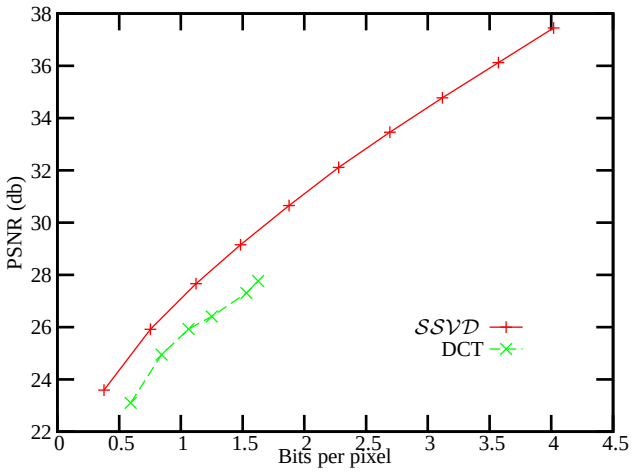


Figure 3. $\mathcal{SSVD}$ vs. DCT

Acknowledgements: We thank V. M. Gadre, Subhasis Chaudhuri and Ravi Kannan for discussions and suggestions.

References

[1] H. C. Andrews and C. L. Patterson. Singular Value Decomposition (SVD) Image Coding. *IEEE Transactions on Communications*, 24:425–432, April 1976.

[2] J. J. Gerbrands. On the relationships between SVD, KLT and PCA. *Pattern Recognition*, 14(6):375–381, 1981.

[3] C. S. M. Goldrick, W. J. Dowling, and A. Bury. Image coding using the singular value decomposition and vector quantization. In *Image Processing And Its Applications*, pages 296–300. IEE, 1995.

[4] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The John Hopkins University Press, 1983.

[5] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall of India, October 2000.

[6] D. Kalman. Singularly Valuable Decomposition: The SVD of a Matrix. *College Mathematics Journal*, 27(1), January 1966.

[7] W. B. Pennebaker and J. L. Mitchell. *JPEG still image data compression standard*. Van Nostrand Reinhold, 1993.

[8] T. Saito and T. Komatsu. Improvement on singular value decomposition vector quantization. *Electronics and Communications in Japan, Part 1*, 73(2):11–20, 1990.

[9] P. Waldemar and T. A. Ramstad. Image compression using singular value decomposition with bit allocation and scalar quantization. In *Proceedings of NORSIG Conference*, pages 83–86, 1996.

[10] J.-F. Yang and C.-L. Lu. Combined Techniques of Singular Value Decomposition and Vector Quantization for Image Coding. *IEEE Transactions on Image Processing*, 4(8):1141–1146, August 1995.