Combining Support Vector Machine Learning With the Discrete Cosine Transform in Image Compression

Jonathan Robinson and Vojislav Kecman

***Abstract*—In this paper, we present a novel algorithm for the application of support vector machine (SVM) learning to image compression. The algorithm combines SVMs with the discrete cosine transform (DCT). Unlike a classic radial basis function networks or multilayer perceptrons that require the topology of the network to be defined before training, an SVM selects the minimum number of training points, called support vectors, that ensure modeling of the data within the given level of accuracy (a.k.a. insensitivity zone ϵ). It is this property that is exploited as the basis for an image compression algorithm. Here, the SVMs learning algorithm performs the compression in a spectral domain of DCT coefficients, i.e., the SVM approximates the DCT coefficients. The parameters of the SVM are stored in order to recover the image. Results demonstrate that even though there is an extra lossy step compared with the baseline JPEG algorithm, the new algorithm dramatically increases compression for a given image quality; conversely it increases image quality for a given compression ratio. The approach presented can be readily applied for other modeling schemes that are in a form of a sum of weighted basis functions.**

***Index Terms*—Image compression, kernel machines, support vector machine (SVM).**

I. Introduction

THE use of neural networks in image compression is not new. Reference [1], for example, describes an algorithm using backpropagation learning in a feedforward network. The number of hidden neurons was fixed before learning and the weights of the network after training were transmitted. The neural network (and hence the image) could then be recovered from these weights. Compression was generally around 8:1 with an image quality much lower than JPEG.

More recently, Amerijckx *et al.* [2] presented a compression scheme based on the discrete cosine transform (DCT), vector quantization of the DCT coefficients by Kohonen map, differential coding by first-order predictor and entropic coding of the differences. This method gave better performance than JPEG for compression ratios greater than 30:1.

The use of support vector machines (SVMs) in an image compression algorithm was first presented in [3]. This method used SVM to directly model the color surface. The parameters of a neural network (weights and Gaussian centers) were transmitted so that the color surface could be reconstructed from a neural network using these parameters.

The compression algorithm presented here follows from the work in [3]. In [3], SVM learning was used to directly model the color surface. In the algorithm presented in this paper, we apply SVM learning to an image after mapping the image into the frequency domain. Compression rate and image quality are much improved as the results will demonstrate.

This paper is organized as follows: Section II gives an overview of SVM learning. As the DCT is the part of the algorithm proposed, Section III discusses the basics of a DCT. Section IV states the problem and introduces the novel compression algorithm. Section V presents the results with comparison to JPEG compression with details on speed of compression. Finally, concluding remarks are given in Section VI.

II. SVM Learning

SVMs have become very popular tools for learning from experimental data and solving various classification, regression and density estimation problems. These novel soft models are dubbed kernel machines too. One way of looking at them may also be as the new learning method for a radial basis function (RBF) neural network. Initially developed for solving classification problems, support vector (SV) techniques can be successfully applied in regression, i.e., for functional approximation problems ([4], [5]). It is this application we will exploit here. Unlike pattern recognition problems, where the desired outputs y_i are discrete values, e.g., Boolean, here we deal with *real-valued* functions. Now, the general regression learning problem is set as follows: the learning machine is given l training data from which it attempts to learn the input-output relationship (dependency, mapping, or function) $f(\mathbf{x})$. A training data set $D = [\mathbf{x}(i), y(i)] \in \Re^n \times \Re$, $i = 1, \ldots, l$ consists of l pairs $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_l, y_l)$, where the inputs $\mathbf{x}$ are n-dimensional vectors $\mathbf{x} \in \Re^n$ and system responses $y \in \Re$, are continuous values. The SVM considers approximating functions of the form

$$f(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) \tag{1}$$

where the functions $\phi_i(\mathbf{x})$ are called basis functions. Equation (1) is an SVM model where N is the number of SVs. In the case of SVM regression, one uses Vapnik's linear loss function with ϵ-insensitivity zone as a measure of the error of approximation

$$|y - f(\mathbf{x}, \mathbf{w})| = \begin{cases} 0, & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \le \epsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \epsilon, & \text{otherwise.} \end{cases} \tag{2}$$

Thus, the loss is equal to zero if the difference between the predicted $f(\mathbf{x}, \mathbf{w})$ and the measured value is less than ϵ. Vapnik's ϵ-insensitivity loss function (2) defines an ϵ tube. (Typical graph of a regression problem as well as all relevant mathematical objects required in learning unknown coefficients w_i are shown in Fig. 1.) If the predicted value is within the tube the loss (error or cost) is zero. For all other predicted points

Manuscript received June 5, 2002; revised March 6, 2003.

The authors are with the School of Engineering, University of Auckland, 92019 Auckland, New Zealand.

Digital Object Identifier 10.1109/TNN.2003.813842

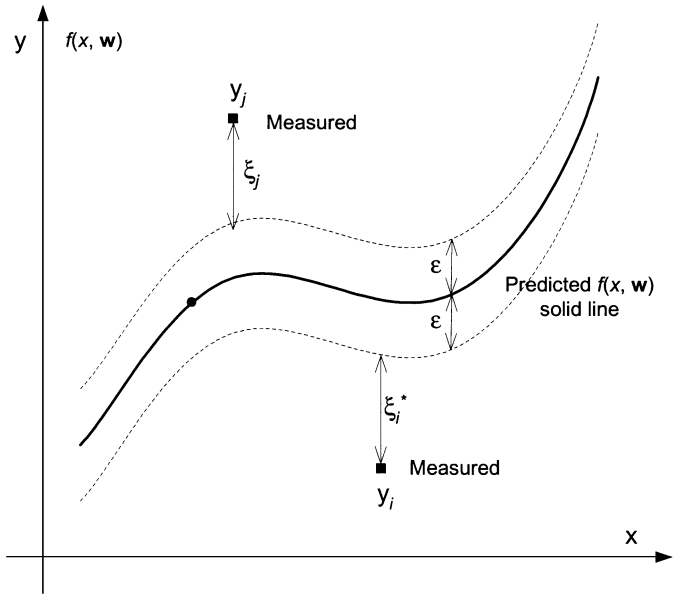

Fig. 1. Parameters used in one-dimensional (1-D) SV regression.

outside the tube, the loss equals the magnitude of the difference between the predicted value and the radius ϵ of the tube. Note that for $\epsilon = 0$, Vapnik's loss function equals a least modulus (a.k.a. Huber's robust loss) function.

In solving regression problems, the SVM performs linear regression in n-dimensional feature space using ϵ-insensitivity loss function. At the same time, it tries to reduce model capacity by minimizing $||\mathbf{w}||^2$, in order to ensure better generalization. All these are achieved by minimizing the following functional:

$$R_{\mathbf{w},\xi,\xi^*} = \frac{1}{2}||\mathbf{w}||^2 + C\left(\sum_{i=1}^{l}\xi_i + \sum_{i=1}^{l}\xi_i^*\right) \tag{3}$$

under constraints

$$y_i - f(\mathbf{x}_i, \mathbf{w}) \leq \epsilon + \xi_i \tag{4a}$$
$$f(\mathbf{x}_i, \mathbf{w}) - y_i \leq \epsilon + \xi_i^* \tag{4b}$$
$$\xi_i \geq 0 \tag{4c}$$
$$\xi_i^* \geq 0 \tag{4d}$$

where ξ_i and ξ_i^* are slack variables shown in Fig. 1 for measurements "above" and "below" an ϵ-tube, respectively. Both slack variables are positive values and they measure the deviation of the data from the prescribed ϵ-tube. Their magnitude can be controlled by penalty parameter C. This optimization problem is typically transformed into the dual problem, and its solution is given by

$$f(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N_{SV}} (\alpha_i^* - \alpha_i)\, G(x_i, x)$$
$$0 \leq \alpha_i^* \leq C,$$
$$0 \leq \alpha_i \leq C \tag{5}$$

where α_i and α_i^* are the Lagrange multipliers corresponding to ξ_i and ξ_i^*, N_{SV} is the number of SVs and $G(x_i, x)$ is the kernel function. Gaussian kernels are used in the compression algorithm detailed in this paper. The constant C influences a tradeoff between an approximation error and the weight vector norm $||\mathbf{w}||$ and it is a design parameter that is typically chosen by the user through cross validation. Here, we worked with $C = \infty$ and all the "softening" of the SVM performance was done by choosing proper value of the ϵ-insensitivity zone. An increase in C penalizes larger errors (large ξ and ξ^*) and in this way leads to an approximation error decrease. However, this can be achieved only by increasing the weight vector norm $||\mathbf{w}||$. At the same time, an increase in $||\mathbf{w}||$ does not guarantee a small generalization performance of a model. Another design parameter which is chosen by the user is the required precision embodied in an ϵ value that defines the size of an ϵ-tube (a.k.a ϵ-insensitivity zone).

The expansion (5) can also be rewritten in a shape more familiar to the neural network community as $f(\mathbf{x}, \mathbf{w}) = \mathbf{G}\mathbf{w}$, where the $\mathbf{G}$ matrix is known as the design (or kernel) matrix and the weight vector (of the kernel expansion) is $\mathbf{w} = \boldsymbol{\alpha}^* - \boldsymbol{\alpha}$.

There are a few learning parameters in constructing SV machines for regression. The two most relevant are the insensitivity zone ϵ and the penalty parameter C that determines the tradeoff between the training error and Vapnik–Chervonenkis (VC) dimension of the model. Both parameters should be chosen by the user. Increase in ϵ means a reduction in requirements on the accuracy of approximation. It decreases the number of SVs leading to data compression too. This will be exploited here in approximating the coefficients of a DCT in a frequency domain. For a detailed mathematical description of SVMs the interested reader is referred to [6]–[8].

III. DCT

The DCT is the fundamental process of the JPEG image compression algorithm [9], [10]. The DCT is a transform that maps a block of pixel color values in the spatial domain to values in the frequency domain. The DCT can operate mathematically in any dimension, however an image is a two-dimensional (2-D) surface so the 2-D DCT transform is used. The 2-D DCT is given by

$$T[i,j] = c(i,j)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} V[x,y] \times \cos\frac{(2y+1)i\pi}{2N} \times \cos\frac{(2x+1)j\pi}{2N} \tag{6}$$

where

$$c(i,j) = \frac{2}{N}, \qquad i \text{ and } j \neq 0$$
$$c(i,j) = \frac{1}{N}, \qquad i \text{ and } j = 0.$$

The DCT is more efficient on smaller images. When the DCT is applied to large images, the rounding effects when floating point numbers are stored in a computer system result in the DCT coefficients being stored with insufficient accuracy resulting in deteriorated image quality. As the size of the image is increased, the number of computations increases disproportionately. For these reasons an image is subdivided into 8×8 blocks. Where an image is not an integral number of 8×8 blocks, the image can be padded with white pixels (i.e., extra pixels are added so that the image can be divided into an integral number of 8×8 blocks). The 2-D DCT is applied to each block so that an 8×8

43	54	54	49	58	45	52	78
42	49	54	49	60	44	55	82
53	47	53	49	56	50	56	96
47	54	49	52	55	54	59	106
49	54	56	47	54	59	61	105
49	52	53	49	68	64	63	95
53	57	67	59	64	67	74	102
58	58	58	60	73	80	81	109

DCT →

← IDCT

488	-81	45	-43	29	-36	15	4
-44	20	-4	-9	5	-2	-6	4
10	5	-13	-6	6	5	-5	5
-11	7	-6	0	-6	-4	-2	3
5	-4	6	-1	-1	-3	-1	-7
2	2	1	-5	0	2	0	-2
-1	-1	0	9	4	-2	1	0
2	3	5	-4	-2	4	5	1

Fig. 2. DCT maps a block of pixel color values to the frequency domain. Note that the magnitude of the coefficients generally increases the nearer they are to the top-left coefficient.

matrix of DCT coefficients is produced for each block. This is termed the DCT matrix. The top left component of the DCT matrix is termed the discrete cosine (DC) coefficient and can be interpreted as the component responsible for the average background color of the block (analogous to a steady DC current in electrical engineering). The remaining 63 components of the DCT matrix are termed the "AC" components as they are frequency components analogous to an electrical ac signal. The DC coefficient is often much higher in magnitude than the AC components in the DCT matrix.

The DCT is illustrated in Fig. 2. Each component in the DCT matrix represents a frequency in the image (the DC component representing a frequency of 0). The further an AC component from the DC component the higher the frequency represented. The magnitude of higher frequency components tends to diminish the higher the frequency represented. Higher frequency components are less visible to the human eye, and it is this property which is exploited in JPEG and in our algorithm as these higher frequency components can be attenuated or removed with little noticeable effect on the quality of the image. Thus the smoothness properties of the SVM can be used to model the DCT coefficients. The trend in the magnitude of the DCT coefficient is visible in Fig. 2 The original image block is recovered from the DCT coefficients by applying the inverse DCT (IDCT), given by

$$T[i,j] = c(i,j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} V[x,y] \cdot \cos \frac{(2y+1)i\pi}{2N} \times \cos \frac{(2x+1)j\pi}{2N} \quad (7)$$

where

$$c(i,j) = \frac{2}{N}, \qquad i \text{ and } j \neq 0$$
$$c(i,j) = \frac{1}{N}, \qquad i \text{ and } j = 0.$$

A. Transformation of the DCT Matrix to 1-D

The elements of the matrix $T[i,j]$ in (6) are mapped using the zig-zag sequence shown in Fig. 3 to produce a single row of numbers. That is a single row of numbers is collected as we follow the zig-zag trail in the DCT matrix. This will produce a row of 64 numbers, where the magnitude tends to decrease as we travel down the row of numbers. Coefficients placed next to each other via the zig-zag mapping tend to be of similar magnitude, thus making the row of coefficients more suitable for generalization by a SVM.

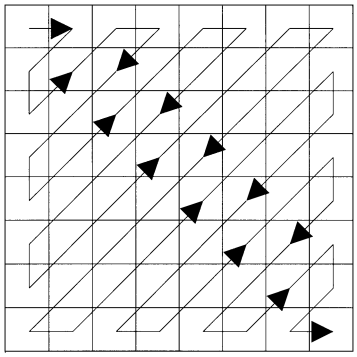

Fig. 3. Zig-zag pattern applied to a block of DCT coefficients to produce a row of 64 coefficients. The importance of each coefficient is proportional to its position in the row.

IV. Combining SVM With DCT

Here we detail a new algorithm for compressing the AC DCT coefficients after the discrete cosine transform has been applied to an image. The algorithm which we call the RKi-1 algorithm, also includes a preparation step to improve the compression efficiency.

A. Statement of the Problem

Fig. 4(a) shows the 1-D plot of the AC coefficients for the matrix shown in Fig. 2 after the zig-zag mapping. This is a plot of a single row of DCT coefficients after the zig-zag mapping has been applied to the block in Fig. 2. The DC component has been removed as it is treated separately leaving the 63 AC coefficients. The 1-D row of DCT coefficients is used as the training data for an SVM. In the discussion of SVMs in Section II attention was drawn to the fact that an SVM will produce the minimum number of SVs required to generalize the training data within a predefined error (the ϵ-insensitivity tube). Thus, we expect that when the row of DCT coefficients are used as training data for the SVM, a lower number of SVs will be required in order to recover the DCT coefficients within the predefined error. This is illustrated in Fig. 5 (a) where the DCT coefficients in Fig. 2 are used as input training data to an SVM. Fig. 5(b) shows the error when the output is compared with the original.

In Fig. 5, there are 63 input training points. With the error (ϵ-insensitivity) set to 0.1, 34 training points were chosen by the SVM as SVs. This can be loosely interpreted as compression of 63:34 or approximately 1.85:1. Although this is not the actual compression ratio of the compressed image, this ratio illustrates the reduction in the number of training points.

Examination of the input data (ie the DCT coefficients) reveals that the magnitudes of the coefficients are generally decreasing as we travel down the row of input data, however the sign (positive or negative) appears to be random. This has the consequence that two coefficients next to each other can be of similar magnitude but opposite sign causing a large swing in the input data.

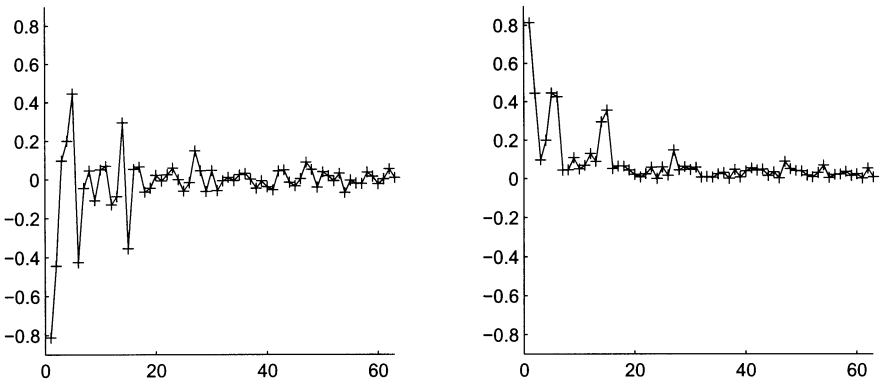

Fig. 4. (a) Typical 8×8 block of DCT coefficients after the zig-zag mapping to a row of numbers. This plot is not easily generalized by a neural network. (b) Absolute magnitude of the same data. This plot, while still not ideal, is better generalized by a neural network.

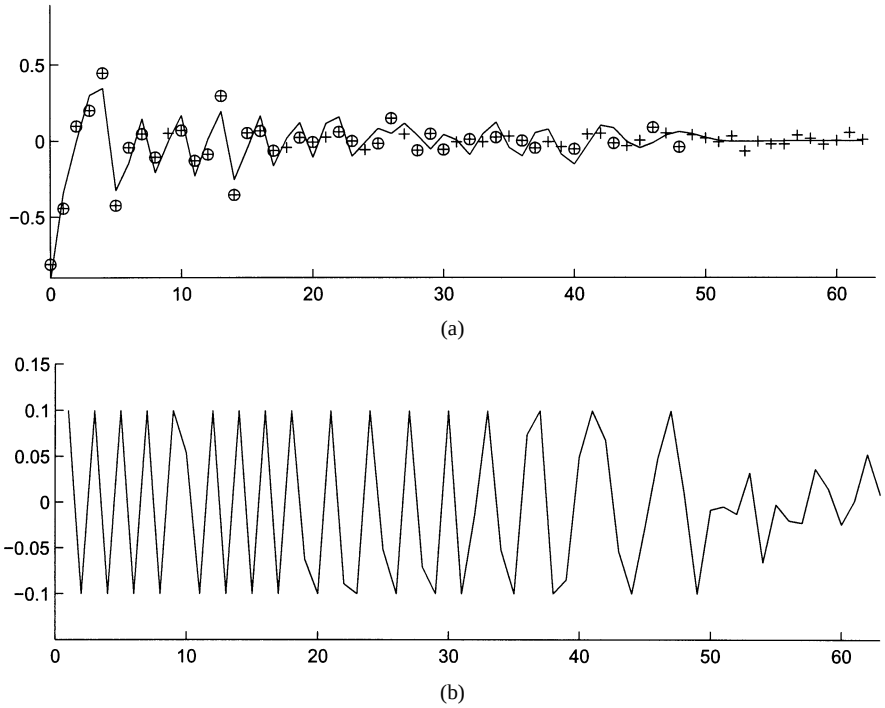

Fig. 5. (a) Real values of the row of DCT coefficients input to the SVM. (b) Error of the output subtracted from the desired input. The maximum allowable error is 0.1.

Fig. 6(a) shows the result when the absolute magnitude of the DCT coefficients is used as input to the SVM. That is, the negative signs are ignored and all inputs are treated as positive. It can be seen that this input data is more suitable for generalization by the SVM as less training points are chosen to be SVs. In this example, the SVM has chosen ten SVs representing a compression of 63:10 or 6.3:1. This is an increase in compression of around 3.5 times over the preceding example when the real values of the DCT coefficients were used as training data. We note also that when the real values were used as training data, the total accumulated error was 452, and when the magnitudes were used the total accumulated error reduced to 271. In this example, by ignoring the sign of the DCT coefficient when used as training data to the SVM, the number of SVs has reduced by $\approx 70\%$ and the accumulated error has reduced by $\approx 40\%$. We conclude that by simply using the magnitude of the input values and ignoring the sign, that the compression is substantially increased and the output is a closer match to the desired output (i.e., the error in the output is reduced). A comparison plot of the input data when both the real values and the absolute magnitudes is shown in Fig. 4.

If we ignore the sign of each DCT coefficient when used as input data to the SVM, we are left with the problem of how to reassign the signs when the DCT coefficients have been recovered. In order to recover the sign, we introduce the inversion bit.

B. Inversion Bit

The inversion bit indicates which of the recovered points on Fig. 4 (b) should be inverted (i.e., multiplied by -1) so that they are negative. The inversion bit is a single "0" or "1." It is the sign of the corresponding input datum. Each

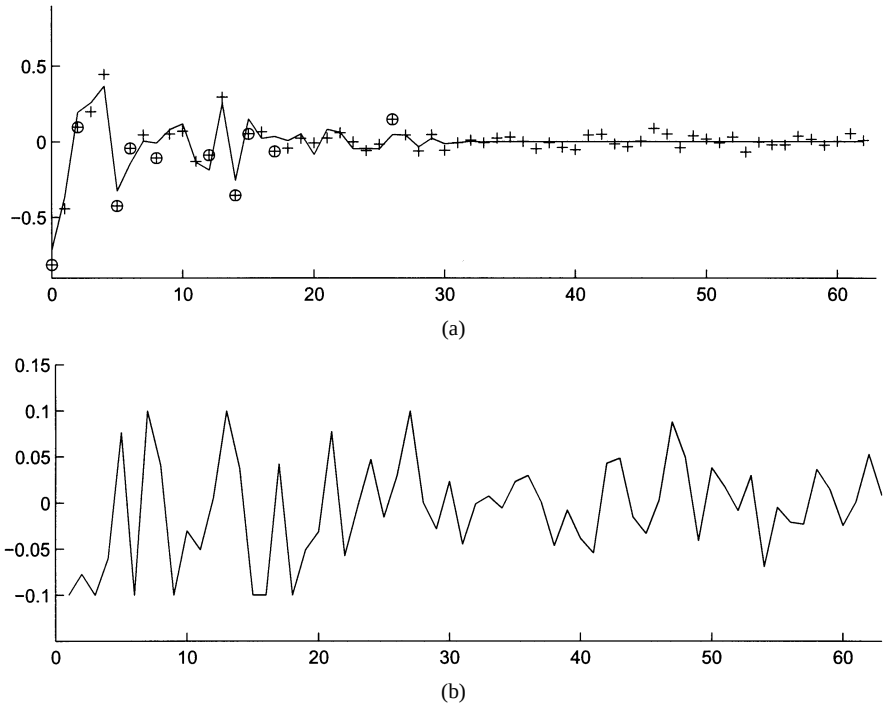

Fig. 6. (a) Absolute magnitude of the row of DCT coefficients input to the SVM. (b) Error of the output subtracted from the desired input. The maximum allowable error is 0.1.

TABLE I
EXAMPLE OF THE INPUT COEFFICIENTS, THE NEURAL-NETWORK PARAMETERS PRODUCED BY SVM LEARNING AND THE APPROXIMATED COEFFICIENTS WHERE THE SVM INSENSITIVITY HAS BEEN SET TO TEN

Original data:

Original Coeffs	-81	-44	10	20	45	-43	-4	5	-11	5	7	-13	-9	29	-36	5
Training Data	81	44	10	20	45	43	4	5	11	5	7	13	9	29	36	5
Inversion No	1	1	0	0	0	1	1	0	1	0	0	1	1	0	1	0

Support vector parameters after SVM learning:

Support Vectors	1	5	6	9	12	14	15
weights	71.3	23.4	18.3	0.6	2.0	5.9	22.0

Approximated (recovered) coefficients:

Original Coeffs	-81	-44	10	20	45	-43	-4	5	-11	5	7	-13	-9	29	-36	5
Recovered Coeffs	-71	-44	13	17	35	-33	-14	3	-1	1	1	-3	-8	20	-26	14
Error	10	0	3	3	10	10	10	2	10	4	6	10	1	9	10	9

input datum has an inversion bit, so for the block shown in Fig. 2 there are 63 inversion bits. These bits together form what we term the inversion number. The inversion number for the matrix of DCT coefficients in Fig. 2 is 110 001 101 001 101 011 101 000 010 010 110 100 011 110 011 000 100 101 011 001 000 After a block has been processed by the SVM, some the recovered DCT coefficients may have a magnitude lower than the maximum error defined for the SVM. If these components had an inversion bit of "1" we can set this to "0" as the sign of coefficients with small magnitude has little effect on the final recovered image. Put another way, inversion bits for very small magnitude DCT coefficients do not contain significant information required for the recovery of the image. For example if we define the ϵ-insensitivity as 0.1, the desired value of a particular DCT coefficient is -0.06, and the recovered value 0.04 then we can set the inversion bit to "0" for this coefficient (rather than "1") because a recovered value of 0.04 is within 0.1 of the original desired value. The importance of this property will be detailed in the implementation section.

V. IMPLEMENTATION

The image is first subdivided into 8×8 blocks. The 2-D DCT is applied to each block to produce a matrix of DCT coefficients. The zig-zag mapping is applied to each matrix of DCT coefficients to obtain a single row of numbers for each original block of pixels. The first term of each row (the DC component) is separated so that only the AC terms are left. Not all the terms in the row of AC coefficients are required since the higher order terms do not contribute significantly to the image. The higher order

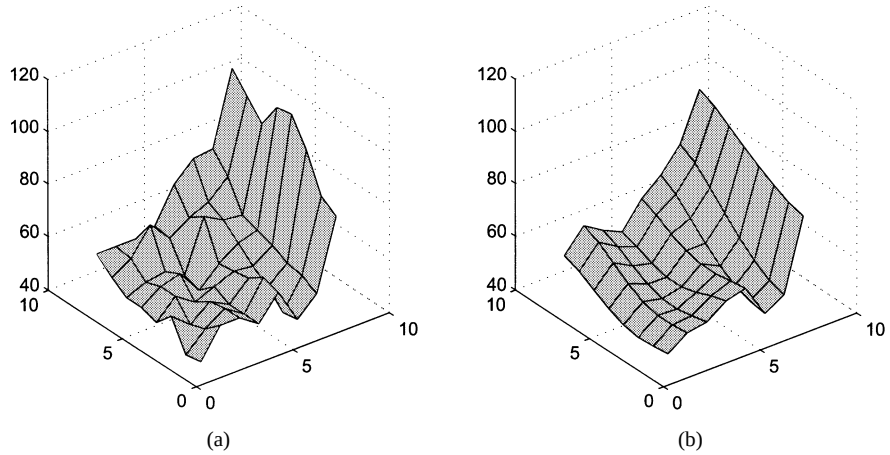

Fig. 7. (a) Original block. (b) Recovered block.

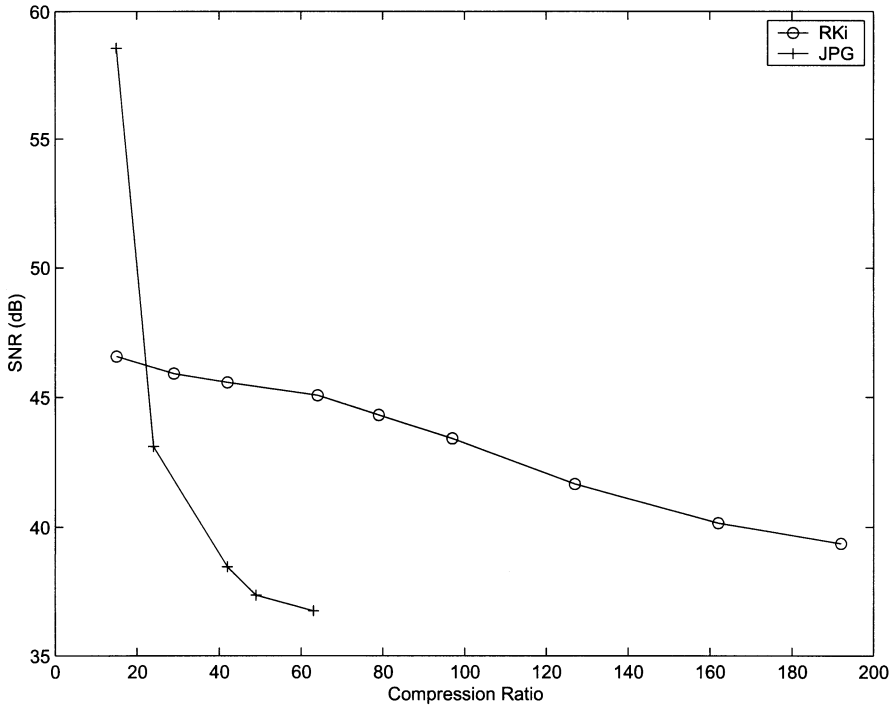

Fig. 8. Compression ratio versus SNR for the new method and baseline JPEG using the 512×512 Lena image.

terms represent higher frequency components of the image and the higher the frequency the less noticeable the frequency component is to the human eye. Because of this we can choose to take only the first values (typically the first 8–16 values). Exactly how many values are taken is a degree of freedom in the algorithm. SVM learning is applied to the absolute values of each row of AC terms as described above and the inversion number for each block is generated. Since we have used Gaussian kernels in the SVM, for each original block the parameters needed to be stored/transmitted are the Gaussian centers (i.e., the SVs), the weights and the inversion number to be able to recover the block. There are no bias terms due to the fact that positive definite Gaussian kernels have been used which do not require bias.

Original Data: Table I gives a numeric example for the image block in Fig. 2. The table shows the first 16 coefficients of the block used as training data to the SVM. Also shown is the corresponding inversion number, the coefficients identified as SVs, the approximated (recovered) coefficients, and the error between each approximated coefficient and the input training coefficient. In this example, the SVM ϵ-insensitivity has been set to 10. Note that the weights have been rounded to one decimal place.

A surface plot of the original 8×8 surface is shown along side the recovered surface in Fig. 7. In this figure, the smoothing effect of the SVM on the recovered block is visible and has the effect of applying a high-frequency filter on the block. In the actual implementation we normalize the pixels in the image between 0 and 1. When the input was not normalized the weights tended to be very large (in the order of 10^8) and a small variation in the weight caused by rounding would significantly deteriorate the image. When the input is normalized, the weights were generally in the range 0–1 and taking the first one or two significant digits of the weight did not adversely affect the quality of the image. A similar approach was suggested in [11]. Normal-

izing the image produces weights that are lower in magnitude (compared with an unnormalized image) and similar in value. This is an important consideration when encoding the data for storage/transmission described in the following section.

A. Encoding Data for Storage

For each block, we need to store weights and SVs. The SVs are the Gaussian centers. In our RKi-1 algorithm we combine the weights with the SVs so that each block has the same number of weights as DCT coefficients. For example, if we chose to retain only the first 16 DCT coefficients (discarding the remaining 47 DCT coefficients), then we have 16 weights for each block. Where a weight has no corresponding SV, we set the value of the weight equal to zero. That is, the only nonzero weights are weights for which a training point has been chosen to be an SV by the SV machine.

The next step is to quantize the weights. By quantizing we mean that we reassign the value of the weight to one of limited number of values. To quantize the weights we find the maximum and minimum weight values (for the whole image) and predefine how many quantization levels to use. The number of quantization levels to use is a degree of freedom in the algorithm. to compute new values for the weights. For example if the minimum weight is 0 and the maximum is 0.9 and we define five quantization levels then the weights can only take the values 0.09, 0.27, 0.45, 0.63, 0.81. Each weight in the image is reassigned to be the closest value of these quantized values.

The inversion bits are combined with the weights in the following way. An arbitrary number is added to all weights (usually "1" added to the absolute value of the smallest weight) so that all weights are positive and nonzero. This arbitrary number must be stored to be able to recover the weights. Each individual weight has an associated inversion bit. The inversion bit is combined with its corresponding weight by making the value of the weight negative if the inversion bit is "1," positive otherwise. Where the weight is not an SV the inversion data is discarded. This introduces a small error when the image is decompressed, but significantly increases compression (see the results section).

The above steps introduce many zero values into the weight data. By setting inversion bits from "1" to "0" when the associated DCT is less than the error, ϵ, we have introduced many more zeros. A combination of Huffman coding and run length encoding (RLE) is used to create a binary image file. The quantized weights and the numbers of zeros between nonzero weight values are Huffman encoded to produce the final binary file.

VI. Results

A. Speed of Compression and Decompression

In RKi-1, compression takes longer than decompression. It was observed that the higher the compression ratio the quicker the algorithm was to compress. The opposite is true for the JPEG algorithm. From a practical user perspective there was little difference in decompression speed between JPEG and our RKi-1 algorithm.

Compression ratios are computed by the following formula:

$$\text{compression} = \frac{\text{image\_height} \times \text{image\_width}}{\text{file\_size in Bytes}} \tag{8}$$

Fig. 9. Subjective comparison of the 512×512 Lena image compressed using both the RKi and JPG algorithms. Note that the JPG algorithm would not compress beyond 64:1.

where file_size is for a binary file containing all parameters and data required to reconstruct the image.

B. Performance

To objectively measure image quality, the signal-to-noise ratio (SNR) is used. The SNR is calculated using

$$\text{SNR} = 20 \log \left[\frac{1}{\text{width} \times \text{height}} \times \frac{\sum_{ij} \text{Original Image}_{ij}}{\sum_{ij} \left(\text{Original Image}_{ij} - \text{Recovered Image}_{ij} \right)} \right]. \tag{9}$$

Results using the benchmark 512×512 Lena image are shown in Fig. 8 in comparison with the baseline JPEG algorithm. The JPEG algorithm performs better than the RKi-1 algorithm for compression ratios up to 22:1 (on this particular image). For compression ratios beyond this, the RKi-1 algorithm produces

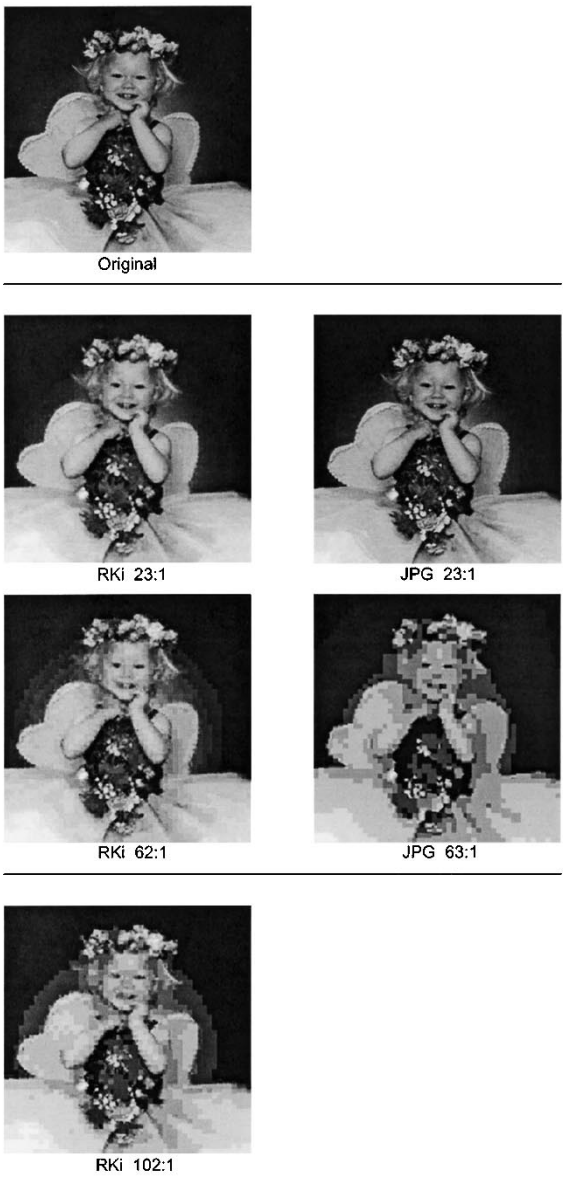

Fig. 10. Subjective comparison of the 400×400 Claudia image compressed using both the RKi-1 and JPG algorithms.

higher quality images for the same compression ratio. While similar results were obtained for other images, the results for the Lena image are detailed as it is the *de facto* standard for comparing image compression algorithms.

The baseline JPEG algorithm could not compress the 512×512 Lena image grater than 64:1. The RKi-1 algorithm achieved a compression ratio of 192:1 and still achieved better image quality than the image compressed using JPEG at 64:1. This is a much better compression ratio than that obtained in [3] where SVM was used to directly model the color surface. When an SVM is used to directly model the surface, compression over 20:1 is not possible without severe deterioration of the quality of the image [3]. Thus, by approximating DCT coefficients of an image rather than directly approximating the color surface, the compression is greatly improved. Fig. 10 shows results for a subjective comparison on the 400×400 Claudia image. The difference in quality between RKi-1 and JPEG is clearly visible at higher compression ratios.

The decompressed images are shown in Fig. 9 for subjective comparison of the RKi-1 algorithm compared with the baseline JPEG algorithm.

VII. Concluding Remarks

We have presented a novel image compression algorithm which takes advantage of SVM learning. The algorithm exploits the trend of the DCT coefficients after the image has been transformed from the spacial domain to the frequency domain via the DCT. SVM learning is used to estimate the DCT coefficients within a predefined error ϵ. The SVM is trained on the absolute magnitude of the DCT coefficients as these values require less SVs to estimate the underlying function. The net result of the SVM learning is to compress the DCT coefficients much further than other methods such as JPEG. The algorithm also defines how the original values are recovered by the introduction of the inversion number. The inversion number allows us to recover the original sign (i.e., positive or negative) of each DCT coefficient so that combined with the magnitude of the coefficient as estimated by the SVM, a close approximation to the original value of the DCT coefficient is obtained in order to reconstruct the image.

We have presented results showing that the new method produces better image quality than the JPEG compression algorithm for compression ratios greater than around 20:1. We have also shown that large compression ratios (192:1) are possible with the new method while still retaining reasonable image quality.

Further work could include applying 2-D SVM learning to the DCT coefficient matrix. This would bypass the need for the zig-zag transformation to produce a 1-D set of DCT coefficients. Further work is also required to explore the results of applying a varying ϵ-tube to the DCT coefficients. In this paper a fixed error (ϵ-insensitivity) has been used. A varying ϵ-insensitivity could be used which reduces in value as a set of DCT coefficients is traveled along.

References

[1] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. Cambridge, MA: MIT Press, 1995.
[2] C. Amerijckx, M. Verleysen, P. Thissen, and J. Legat, "Image compression by self-organized Kohonen map," *IEEE Trans. Neural Networks*, vol. 9, pp. 503–507, May 1998.
[3] J. Robinson and V. Kecman, "The use of support vectors in image compression," *Proc. 2nd Int. Conf. Engineering Intelligent Systems*, June 2000.
[4] H. Drucker, C. J. C. Burges, L. Kaufmann, A. Smola, and V. Vapnik, *Support Vector Regression Machines*. Cambridge, MA: MIT Press, 1997, Advances in Neural Information Processing Systems, pp. 155–161.
[5] V. Vapnik, S. Golowich, and A. Smola, *Support Vector Method for Function Approximation, Regression Estimation and Signal Processing*. Cambridge, MA: MIT Press, 1997, vol. 9, Advances in Neural Information Processing Systems.
[6] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.
[7] ——, *Statistical Learning Theory*. New York: Wiley, 1998.
[8] V. Kecman, *Learning and Soft Computing: Support Vector Machines, Neutral Networks and Fuzzy Logic Models*. Cambridge, MA: MIT Press, 2001.

[9] J. Miano, *Compressed Image File Formats*. Reading, MA: Addison-Wesley, 1999.
[10] "Digital Compression and Coding of Continuous-Tone Still Images," Amer. Nat. Standards Inst., ISE/IEC IS 10918-1, 1994.
[11] J. Jiang, "Image compression with neural networks—A survey," *Signal Processing: Image Communication*, vol. 14, 1999.
[12] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1993.
[13] C. Lawson and R. Hanson, *Solving Least Square Problems*. Englewood Cliffs, NJ: Prentice-Hall, 1974.
[14] V. Cherkassky and F. Mulier, *Learning From Data: Concepts, Theory and Methods*. New York: Wiley, 1998.