

Novel Adaptive Image Compression

Dimitrios Charalampidis
Electrical Engineering Department
University of New Orleans
2000 Lakeshore Dr.
New Orleans, LA 70148

Phone: (504) 280-7415, Fax: (504) 280-3950
Email: dcharala@uno.edu

Abstract: *This paper introduces a novel adaptive technique for image compression. The idea is based on adaptive rather than fixed transforms for image compression. One such class of algorithms has been previously based on neural networks (NN). Comparisons are presented in order to show the superiority of the proposed technique over NN architectures. It is shown that the proposed technique results in higher image quality for a given compression ratio than existing NN image compression schemes. It is also shown that training of the proposed architecture is significantly faster than that of NN-based techniques and that the number of learning parameters is small. This allows the coding process to include adaptation of the learning parameters, thus, compression does not depend on the selection of the training set as in NN structures. The overall technique, including some additional lossless coding steps, is compared to JPEG in terms of two measures, namely, PSNR for a given compression ratio, and computational complexity. Certain advantages of the proposed technique over JPEG, including higher PSNR for high compression ratios, are presented.*

1. Introduction

Today, a growing number of industrial and commercial applications require efficient compression of still images for storage and transmission over channels. Despite the existence of image compression standards such as JPEG and the currently developed JPEG2000, further study of efficient image compression schemes is needed. Existing standards are mostly based on fixed transforms. Some adaptability is introduced in multi-resolution algorithms including wavelet-based approaches such as JPEG2000. One branch of investigations that use adaptive techniques has evolved with the involvement of neural networks (NN) in image compression [1]-[8]. Advantages of NN over JPEG may

include robustness under noisy conditions, and simple decoding. Nevertheless, compression using NN suffers from several drawbacks, including slow compression, high computational complexity, moderate compression ratio (CR), and moreover, the reconstructed image quality is training-data dependent. Prior techniques include single-structure and parallel-structure NN architectures. Parallel-structures result in higher decoded image quality than single-structures for a given CR, however, they have higher coding time requirements due to the utilization of multiple NNs.

This paper introduces a different approach to adaptive image compression that overcomes the aforementioned problems. In particular, the proposed architecture consists of a cascade of simple adaptive linear units. The similarity between the proposed and parallel-NN implementations is that image-blocks may be encoded by different parts of the architecture. It is possible, if one desires to model the proposed architecture as a single multistage NN with a single node at each hidden layer. Then, an image-block is encoded using only a subset of the stages. Therefore, the total number of learning parameters is small, and estimation of the learning parameters is considerably faster than that of NN techniques. These two qualities allow training to be part of coding. As a result, the architecture is formed based on the image to be encoded, and coding is not dependent on training data.

The proposed technique adapts to the image content and calculates a set of transforms that code the image in a block-manner similar to JPEG. Transforms such as DCT and wavelets have been previously used for image recognition and characterization. Wavelets have an advantage over DCT due to their multi-resolution properties. Nevertheless, these transforms are fixed and not formed based on the image content. The adaptive nature of the proposed algorithm allows extraction of information from image blocks through the calculation of sub-optimal transforms and their associated coefficients.

This paper is organized as follows. Section 2 presents some background of existing image compression techniques. Section 3 introduces the proposed adaptive architecture. Results in section 4 compare the performances of the proposed method, and NN techniques and JPEG in terms of computational complexity and Peak Signal to Noise Ratio. Finally, section 5 closes with some concluding remarks.

2. Background

This section presents a sort description of image compression techniques, including the JPEG standard and NN architectures.

2.1 The JPEG standard

JPEG [11],[12] is a standard that has been extensively used for many years due to its simplicity, and generally good performance for a variety of applications. Recently, a wavelet-based extension of JPEG, namely JPEG2000 [13], is becoming a standard.

Although there are several descriptions of the JPEG algorithm in the literature, including the standard documentation itself, a short description is presented here for completeness. The JPEG encoding algorithm consists of the following steps:

- The image is split into 8×8 non-overlapping blocks.
- The two-dimensional DCT is calculated for each block.
- The DCT coefficients for each block are ordered in a zigzag manner. As a result, coefficients corresponding to lower frequencies are ordered first.
- The DCT coefficients are linearly quantized.
- Special attention is given to the DC component values of each block which are differentially encoded.
- Insignificant coefficients are ignored (set equal to zero).
- Since not all DCT coefficients are used, a delimiter is used to indicate the end of the required coefficient sequence in a block.
- Runlength coding is used to compress the large number of consecutive zero values.
- Huffman or arithmetic coding is used as an additional lossless compression step.

Decoding in JPEG is the reverse process of coding as it is described above.

Although the core of the proposed technique is of

adaptive nature and could be better compared to NN architectures, it is in its overall structure more similar to JPEG as it will be described later. It can be portrayed as a JPEG-like algorithm with an adaptive-based instead of a fix-based transform core.

2.2 NN for image compression

NN-based image compression architectures have been used in the literature [1]-[8], but consideration has been given mostly to the image quality aspect of compression for a given CR. On the other hand, training speed has not been extensively studied due to the inherently slow training process of most NN architectures. This necessitates that training is performed on images other than the ones to be encoded. Nevertheless, investigators have shown special interest in the development of NN techniques because of some advantages of NN over other image compression techniques such as JPEG, which include simple decoding and robustness to noise [9]. Two major categories of NN-based compression schemes are the single-structure NN and parallel-structure NN architectures. Some background on a single and a parallel structure NN schemes is presented in sections 2.1 and 2.2 respectively.

2.2.1 Single NN-based image compression

One of the first approaches to NN-based compression is presented in Cottrell et al [1], where a single-structure NN approach is introduced for small images. Another single-structure approach (along with a parallel NN approach) is introduced in Carrato et al [2]. The image is split into J blocks $\{\mathbf{B}_j, j = 1, 2, \dots, J\}$ of size $M \times M$ pixels. The pixel values in each block are rearranged to form a M^2 length pattern $\mathbf{C}_j = \{C_{1,j}, C_{2,j}, \dots, C_{M^2,j}\}$, where $j = 1, 2, \dots, J$ ($C_{i,j}$ is the i^{th} element of the j^{th} pattern). It is common practice to normalize the input patterns using the transformation $\mathbf{P}_j = f(\mathbf{C}_j) = (\mathbf{C}_j - m_{C_j})/\sigma_{C_j}$ where m_{C_j} and σ_{C_j} are, respectively, the average and standard deviation of \mathbf{C}_j . Patterns \mathbf{P}_j are used in the training phase of the NN as both inputs and outputs. The NN consists of three layers {input/hidden/output} with number of nodes M^2, H, M^2 , respectively.

The NN acts as a coder/decoder. The coder consists of the input-to-hidden layer weights $v_{i,k} \{i = 1, \dots, M^2 \text{ and } k = 1, \dots, H\}$, and the decoder consists of the hidden-to-output layer weights $w_{k,m} \{k = 1, \dots, H \text{ and } m = 1, \dots, M^2\}$. In the definitions above, $v_{i,k}$ represents the weight from the i^{th} input node to the k^{th} hidden node, and $w_{k,m}$ represents the weight from the k^{th} hidden node to the m^{th} output node. Compression is achieved due to the transformation of patterns \mathbf{P}_j , through the $v_{i,k}$ weights, by

setting the number of hidden nodes H smaller than the input pattern length M^2 ($H < M^2$). The coding product associated to the j^{th} pattern \mathbf{P}_j is the hidden layer output $\mathbf{O}_j = \{O_{1,j}, O_{2,j}, \dots, O_{H,j}\}$. The set $\{\mathbf{O}_j, m_{C_j}, \sigma_{C_j}\}$ together with weights $w_{k,m}$ is sufficient for reconstructing an approximation \hat{C}_j of the original pattern C_j in the decoding phase. Considering the overhead $\{m_{C_j}, \sigma_{C_j}\}$ due to the normalization function f , the CR achieved is $M^2:(H+2)$. The encoding/decoding processes are shown in Figures 1(a) and 1(b), respectively.

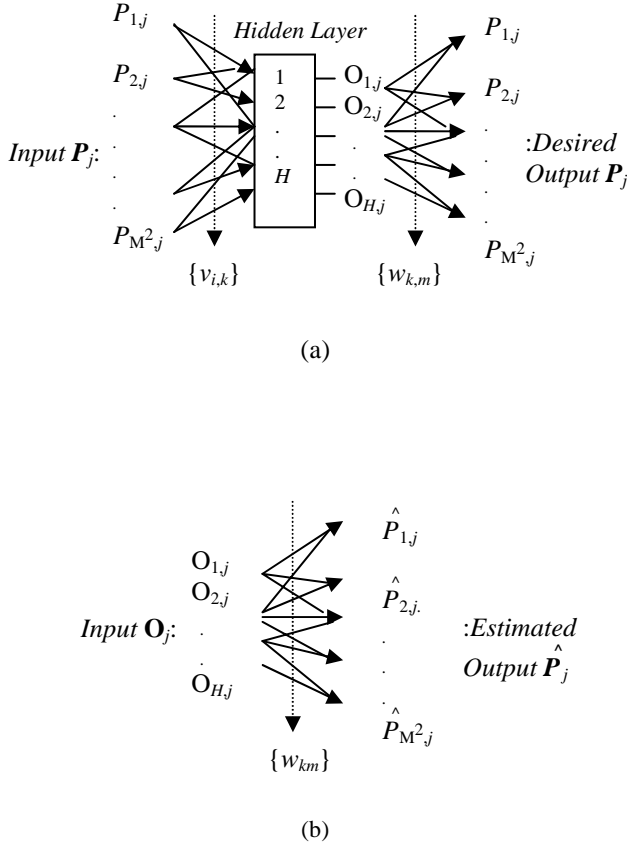


Figure 1. Single neural network architecture with a single stage for image compression (a) Training phase (b) Decoding phase.

2.2.2 Parallel NN-based image compression

Recently, a parallel NN structure has been introduced [3]. Four networks $\{\text{NET}_k, k = 1, 2, 3, 4\}$ with

different number of hidden nodes (4, 8, 12, and 16) are used. Each NN is trained similarly to the NN in section 2.1. The goal is to achieve CR = 8:1.

The coding procedure consists of two phases. In the first phase, each pattern is associated to a NET_k . It is assumed that the larger the number of hidden nodes of the NN to which a pattern is assigned, the smaller the associated error $e_j^2 = (\mathbf{P}_j - \hat{\mathbf{P}}_j) (\mathbf{P}_j - \hat{\mathbf{P}}_j)^T$ between the original \mathbf{P}_j and estimated patterns $\hat{\mathbf{P}}_j$. On the other hand, a large number of hidden nodes results in low CR. Initially, patterns are assigned to NNs so that the CR is as close as possible to the predefined value 8:1.

The second phase is an iterative procedure. At each successive iteration, the goal is to reduce the total error $E^2 = \sum e_j^2$ without changing the CR. Let $de_{j,k}^2$ be the error reduction caused due to reassigning pattern \mathbf{P}_j from NET_k to NET_{k+1} . Then the goal is achieved by reassigning a pair of patterns. Pattern \mathbf{P}_{j_1} is reassigned from NET_{k_1} to NET_{k_1+1} if the reassignment causes a maximum error decrease $de_{j_1,k_1}^2 = \max_k(de_{j_1,k}^2)$, and pattern \mathbf{P}_{j_2} is reassigned from NET_{k_2} to NET_{k_2-1} if this results in a minimum error increase $de_{j_2,k_2}^2 = \min_k(de_{j_2,k-1}^2)$. Iterations continue as long as the error E^2 decreases.

This parallel architecture has the advantage of providing better image quality for a given CR than previous methods [4]-[7] in terms of error E^2 , including both single and parallel structures presented in Carrato et al [2]. Furthermore, coding is faster than previous parallel architectures. Nevertheless, training is still significantly slow due to the use of multiple networks. Moreover, the total number of weights is large. Thus, training cannot be part of the coding process; otherwise, it cannot be performed in real-time. Thus, the compression quality depends on the training data and their similarity to the test images.

3. Proposed adaptive algorithm

This section introduces an image compression method based on a cascade of adaptive transforms. The proposed method exhibits a fast training phase which is included in the encoding process. Thus, it achieves independence of the training data which makes it appropriate for general image compression applications. The encoding and decoding phases of the proposed technique are depicted in Figures 2 and 3 respectively. A detailed description of each block is presented next.

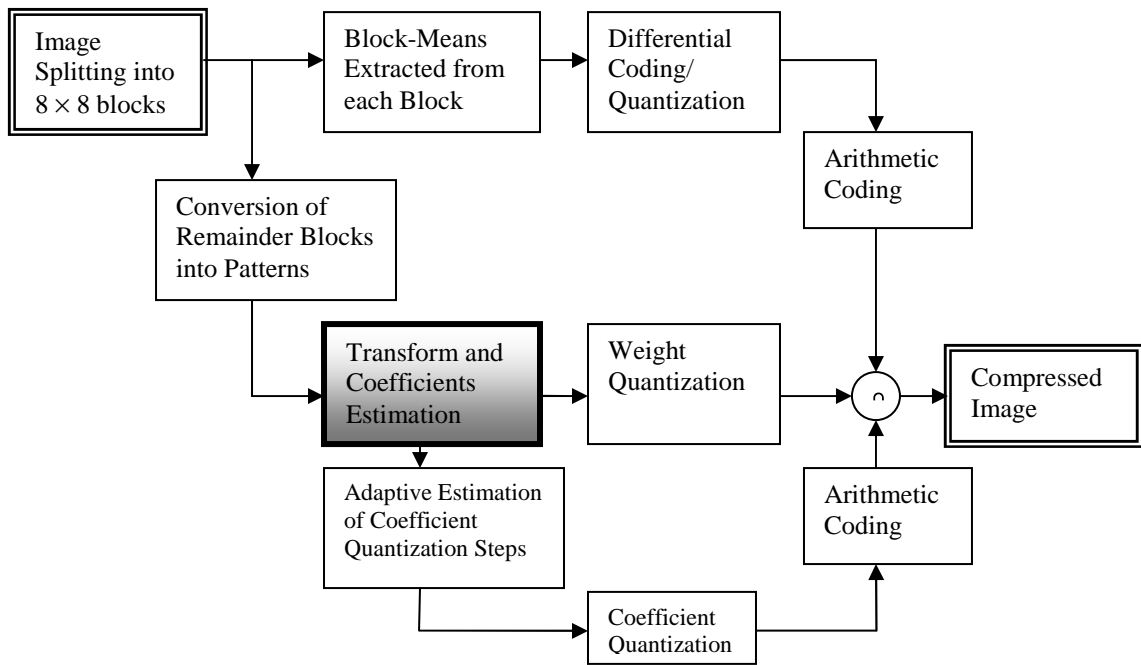


Figure 2. Encoding Phase of the proposed architecture

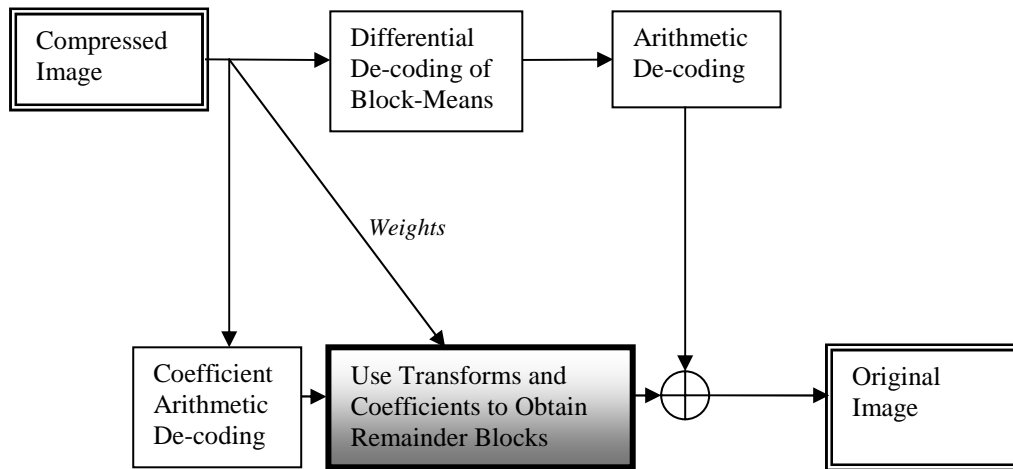


Figure 3. Decoding Phase of the proposed architecture

3.1 Encoding process

In this section, the details about each block in the encoding phase of Figure 1 are presented.

Image splitting into blocks: The image is split into 8×8 non-overlapping image blocks. This process is similar to

the one presented in section 2.2, and it is the same as the one used in JPEG. Different image block sizes could be considered. Larger processing blocks could provide a smaller number of transform coefficients but would increase the overhead due to the number of transform weights that need to be stored. Block size 8×8 was found to be appropriate for the tested images.

Block means (DC) extracted from each block: Since the mean (DC value) of each block j , denoted as m_{C_j} , is expected to contain a significant amount of information, it is extracted and treated (coded) separately as it is described in the next processing block.

Differential coding/quantization of mean values: The mean values are expected to be relatively similar in consecutive blocks, thus they are differentially encoded: the difference of consecutive mean values $m_{C_{j+1}} - m_{C_j}$ is linearly quantized. The number of quantization levels can be variable based on the expected compression ratio.

Conversion of remainder blocks into patterns: The non-overlapping 8×8 blocks are converted into patterns $C_j = [C_{1,j}, C_{2,j}, \dots, C_{M^2,j}]$ as described in section 2.2. Since the block mean value is extracted from each block, the remainder patterns are defined as:

$$P_j = f(C_j) = (C_j - m_{C_j}) \quad (1)$$

The transform and coefficient estimators of the proposed adaptive technique will be referred to as *units*. Patterns P_j are used as inputs/desired-outputs in order to train the first unit.

Weights and coefficients estimation: This is the core of the proposed algorithm. In essence, it is an alternative for the DCT which is the heart of JPEG. This step involves the simultaneous estimation of transforms (weights) and the associated coefficients.

The coefficient estimated from the first unit corresponding to the j^{th} pattern is denoted as $O_{j,k}$, $k = 1$. The first unit's estimated weight vector is denoted as

$$\mathbf{W}_k = [W_{1,k}, W_{2,k}, \dots, W_{M^2,k}], k = 1, \quad (2)$$

The vector \mathbf{W}_k defines the k^{th} transform. Accordingly, element $W_{i,k}$ is the i^{th} weight associated to the k^{th} unit. The coefficient $O_{j,1}$ and the weight vector \mathbf{W}_1 are required in the decoding process.

The first unit's error patterns are denoted as

$$\mathbf{e}_{j,1} = [e_{1,j,1}, e_{2,j,1}, \dots, e_{M^2,j,1}] \quad (3)$$

They are defined as the difference between the original P_j and estimated patterns \hat{P}_j at the output of the first unit, after training is complete. The element $e_{i,j,k}$ is the i^{th} element of the j^{th} error pattern at unit k . If the set of error patterns at the output of unit k is defined as \mathfrak{R}_k then only a subset of these error patterns is used as input/output to train the next unit. This subset \mathfrak{R}_k^s consists of S error patterns $\mathbf{e}_{j,k}^s$, $j = 1, \dots, S$, whose square sum is larger than a specified threshold Q : $\mathfrak{R}_k^s = \{ \mathbf{e}_{j,k}^s \in \mathfrak{R}_k, \mathbf{e}_{j,k}^s \mathbf{e}_{j,k}^{s,T} > Q \}$.

Again, the second unit's coefficients, denoted as $O_{j,2}$, and the weight vector \mathbf{W}_2 are stored to be used in the decoding process.

Similarly, the second unit's error patterns, $\mathbf{e}_{2,j}$, are defined as the difference between the actual $\mathbf{e}_{j,1}^s$ and the estimated error patterns $\hat{\mathbf{e}}_{j,1}^s$ at the second unit's output. Only a subset of the new error patterns $\mathbf{e}_{j,2}$ is used as input/desired-output to train the third unit. The procedure of adding/training units is repeated for as long as the CR does not exceed a specific target. There is an additional overhead per block indicating how many units encode that block. It is important to note that since only a subset of error patterns trains each unit, the number of coefficients $O_{j,k}$ per unit k is variable. This allows assignment of image-blocks with larger estimation error to more units, while keeping the same CR.

The threshold Q is based on the first unit's error patterns $\mathbf{e}_{j,1}$ and the CR. More specifically,

$$Q = a \left[\frac{1}{M^2} \sum_{i=1}^{M^2} \text{var}(e_{i,j,1}) \right] CR \quad (4)$$

The justification for the threshold definition in equation (4) is as follows. A small threshold indicates an expectation for a small coding error. First, the threshold Q is proportional to the desired CR (assuming that there is no additional lossless coding process). A small CR promises a small coding error, therefore, the threshold can be set low. Second, the threshold is proportional to average variance (AV) of the error patterns between the original and the encoded images (using only the first unit). The AV (content of the square bracket in equation (4)) is a "similarity" measure between the error patterns. A small AV indicates that the error patterns may be relatively similar throughout the image-blocks. As a result, the additional adaptive units are expected to be able to produce a small coding error, thus the threshold can be set low. Finally, parameter a is a constant which was set fixed and equal to 1.2 for all experiments.

The advantage of the adaptive technique over parallel NN architectures is that the total number of weights is significantly smaller while the number of coefficients, as compared to hidden node outputs, is variable. Furthermore, the training time requirements are low due to the units' low computational complexity. The algorithm converges in 4-5 iterations by repeatedly applying the set of equations:

$$O_{j,k} = \mathbf{W}_k \mathbf{T}_{j,k}^T / \mathbf{W}_k \mathbf{W}_k^T \quad (5)$$

$$\mathbf{W}_k = \sum_j O_{j,k} \mathbf{T}_{j,k} / \sum_j O_{j,k}^2 \quad (6)$$

where $T_{j,k}$ is equal to P_j if $k = 1$, and $e_{j,k-1}^s$ otherwise.

Equation (5) gives the unit's optimum coefficients $O_{j,k}$ given the unit's weights. This is the result of minimizing the sum of square errors (SSE) between input and output patterns

$$SSE_k = \sum_j (T_{j,k} - \hat{T}_{j,k}) (T_{j,k} - \hat{T}_{j,k})^T, \quad (7)$$

where $\hat{T}_{j,k} = O_{j,k} \mathbf{W}_k$, for unit k with respect to $O_{j,k}$. Similarly, equation (6) gives the unit's optimum unit's weights given the coefficients $O_{j,k}$. This is the result of minimizing $SSE_{j,k}$ with respect to \mathbf{W}_k . The conditions from which equations (5) and (6) are derived are shown next:

$$\frac{\partial SSE_k}{\partial O_{j,k}} = 0, \quad \text{and} \quad \frac{\partial SSE_k}{\partial \mathbf{W}_k} = 0 \quad (8)$$

Since only the transform weights \mathbf{W}_k and the associated coefficients $O_{j,k}$ are needed in the decoding process, it is imperative to obtain the optimum set $\{\mathbf{W}_k, O_{j,k}\}$ for each transform. The algorithm based on equations (5) and (6) directly attempts to find suboptimum values for both \mathbf{W}_k and $O_{j,k}$. The weights and coefficients estimation block is shown in Figure 4.

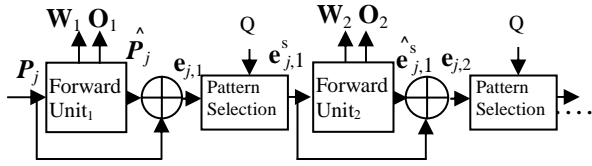


Figure 4. Weights and coefficients estimation block

Weight quantization: After adaptation, the weights are linearly quantized using 256 quantization levels (one byte per weight) for weight values in the interval $[-1, 1]$. Before quantization, the transform weights are normalized so that the maximum absolute value equals 1 as shown in the following equation:

$$\mathbf{W}_k = \mathbf{W}_k / \max_i (|W_{i,k}|) \quad (9)$$

Consecutively, the corresponding coefficients are scaled in order to leave the product $O_{j,k} \mathbf{W}_k$ unchanged

$$O_{j,k}^{\text{scaled}} = O_{j,k} \cdot \max_i (|W_{i,k}|) \quad (10)$$

Adaptive estimation of coefficient quantization steps/ Coefficient quantization: The coefficients estimated from the iterative procedure of equations (5) and (6), and rescaled in equation (10) are linearly quantized. An

important observation is that, since the algorithm estimates the “best” coefficients/weights pair at any given stage k , the coefficients are, in general, more likely to obtain lower absolute values as the number of transforms increases (higher k values). Thus, it is expected that there will be a relatively high concentration of coefficient values around 0. It is desired to use as less quantization levels as possible without significantly increasing the quantization error. Depending on the desired image quality, the coefficients can be quantized using more or less levels. In this work, 32 quantization levels (5 bits) are considered as the maximum number of levels. Moreover, quantization can be performed with the perspective that an additional lossless coding step will be used. Thus, coefficients are quantized as:

$$O_{j,k}^{\text{quant}} = \text{round} \left(\frac{15 \cdot O_{j,k}^{\text{scaled}}}{\Delta \cdot \max_i (|O_{j,k}^{\text{scaled}}|)} \right) \quad (11)$$

Parameter Δ takes values greater or equal to 1, and it is used to refine the quantization levels. A large Δ value will force the quantized coefficients $O_{j,k}^{\text{quant}}$ to assume small values. This is helpful in the arithmetic coding step of the coefficients presented next, because it places a significant number of coefficient values around zero. Since arithmetic coding is based on the symbols' probability of occurrence, higher compression can be achieved if certain coefficient values are more probable. The Δ value is estimated using an adaptive mechanism. More specifically, parameter Δ is adapted as follows:

$$\Delta^t = \begin{cases} (1 + \mu^t) \Delta^{t-1}, & \text{if } SSE\{\Delta = \Delta^t\} < \varepsilon SSE\{\Delta = 1\} \\ (1 - \mu^t) \Delta^{t-1}, & \text{if } SSE\{\Delta = \Delta^t\} > \varepsilon SSE\{\Delta = 1\} \end{cases} \quad (12)$$

and

$$\mu^t = \begin{cases} \mu^{t-1}/2, & \text{if } \{SSE\{\Delta = \Delta^t\} > \varepsilon SSE\{\Delta = 1\}\} \text{ AND} \\ & \{SSE\{\Delta = \Delta^{t-1}\} < \varepsilon SSE\{\Delta = 1\}\} \\ \mu^{t-1}, & \text{otherwise} \end{cases} \quad (13)$$

where the notation $SSE\{\Delta = x\}$ defines the error as in (7) but $O_{j,k}$ and \mathbf{W}_k are substituted by their quantized versions for $\Delta = x$. Parameter ε is a small number indicating the percentage of additional quantization error that adjustment of parameter Δ is allowed to introduce over the case where $\Delta = 1$. Parameter μ is the learning parameter at iteration t . In this work, ε was selected equal to 0.01, and $\mu^0 = 0.1$.

It should also be mentioned that equation (11) results in a maximum of 31 levels. The 32nd level is dedicated to the block delimiter.

Arithmetic coding for block mean values: The basic arithmetic coding scheme is used for lossless coding of the quantized mean differences $m_{C_{j+1}} - m_{C_j}$.

Arithmetic coding for coefficients: The coefficients extracted from each block are placed in order and separated from the next block coefficients by a delimiter (the 32nd unused level of the quantization scheme). The basic arithmetic coding scheme is used for lossless coding of the coefficient stream. The symbol histogram can be estimated globally for small, and locally for large images.

As mentioned earlier, the nature of the algorithm and the additional adaptive quantization level estimation allow a large number of coefficient values to be concentrated around the 0 value. This helps arithmetic coding to perform significant lossless compression.

3.2 Decoding process

Each image block is decoded using the reverse of the encoding process as shown in Figure 3. More specifically, the operations performed by the highlighted block of Figure 3 is described next and shown in more detail in Figure 5.

The set $\{O_{j,k}, \mathbf{W}_k\}$, considering all units k , is used to encode that block. For instance, the first unit produces an estimate of patterns $\hat{P}_j = O_{j,1} \mathbf{W}_1^T$ and the second unit an estimate of the first unit's error patterns $\hat{e}_{j,1}^s = O_{j,2} \mathbf{W}_2^T$. The decoded block is obtained from summing of all those estimates and the block average m_{C_j} .

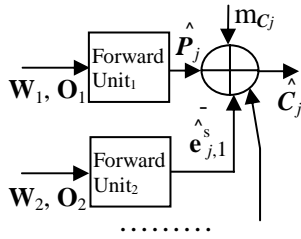


Figure 5. Detailed decoding block (:highlighted block in Figure 3).

3.3 Transform interpretation

This section provides an interpretation of the estimated transforms and presents similarities between the proposed method, NN techniques and JPEG.

Figure 6(a) shows the “Baboon” image. Figures 6(a)-(e) present some of the transforms found using the proposed algorithm. In order to provide a meaningful spatial interpretation, the weight vectors have been converted into 8×8 size matrices, and presented as surfaces. It should be mentioned that similarly, JPEG uses 8×8 DCT transforms that could also be represented as different surfaces.

For instance, Figure 6(b) presents the first transform

($k = 1$). It can be observed that it represents a surface with an oriented slope. This transform is appropriate for edges with that orientation. The algorithm found that this is the “best” transform in order to approximate the 8×8 blocks used to train the first unit. A local reconstructed image component based on the first transform is presented in Figure 7(b). This is the location identified by the small square in Figure 6(a). The zoomed original image square is presented in Figure 7(a). It can be observed that there is an edge (dashed line) and this transform partially attempts to approximate it.

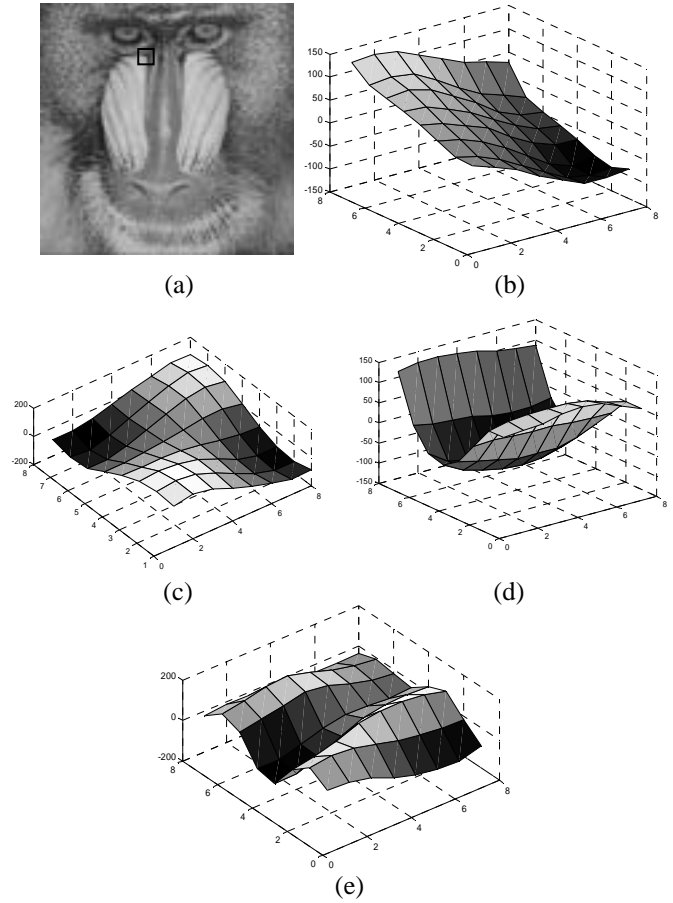


Figure 6. (a) Baboon Image, and (b),(c),(d),(e) Some of the resulted transforms.

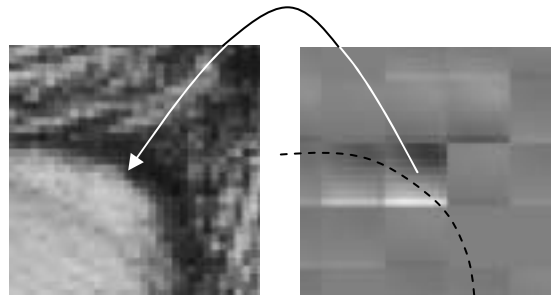


Figure 7. (a) Zoomed location in the image, (b) corresponding approximation due to the first transform.

Similarly, Figure 8(a) shows a textural image. Figures 6(b) and (c) present two of the transforms found using the proposed algorithm. For instance, the transform presented in Figure 8(b) was the “best” transform found by the algorithm. This is an expected result since the textural image consists of almost vertical lines.

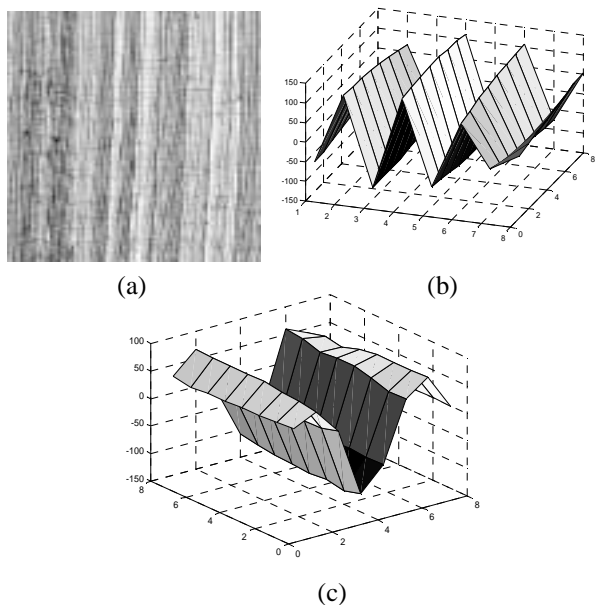


Figure 8: (a) Texture Image, and (b),(c) Two of the resulted transforms.

In general, the proposed technique, JPEG, and NN techniques have something in common. They all use transforms and associated coefficients to get an approximated version of the original image. The transform used in JPEG is fixed (DCT), while the proposed technique and NN techniques use adaptive transforms (one can think of the NN hidden node outputs as the “coefficients” and the weights emanating from the hidden to the output layer as the “transforms weights”). As mentioned earlier, NN architectures need to be trained with images other than the ones to be encoded. Thus, although the transforms are found using an adaptive process, they are still fixed in the test phase (actual compression phase).

4. Experimental results

This section presents results which compare the proposed method with NN-based methods and JPEG. The comparisons with NN do not include an additional lossless step. They are presented to illustrate the

advantages of the proposed technique, and to justify why it is a better choice for being used in JPEG-like algorithms than NN.

4.1 Comparisons with NN architectures

This section presents comparisons between the single-structure and parallel-structure architectures presented in section 2, and the proposed method, in terms of Peak Signal to Noise Ratio (PSNR) and computational complexity. These comparisons do not include any lossless compression components. The CR is defined as the number of pixels (bytes) in the original image over the total number of coefficients (or hidden node outputs) assuming that each coefficient was quantized using 256 levels (one byte).

4.1.1 Comparisons in terms of PSNR

Table 1 compares the single-structure NN and the proposed method in terms of PSNR for 4 different images. The single-structure NN is both trained and tested on the same image to avoid dependence of the results on training data. Although this is impractical for real applications due to high training time requirements, it is useful for comparison purposes. Even in this case, Table 1 shows that the proposed architecture gives higher PSNR for all tested images and for three different CRs (4:1, 8:1, 16:1).

CR	Lena		Baboon		Peppers		Girl	
	Single Structure	Cascade	Single Structure	Cascade	Single Structure	Cascade	Single Structure	Cascade
16:1	29.2	31.8	21.6	21.8	28.8	31.7	28.9	30.3
8:1	31.9	35.8	22.9	23.4	31.9	35.1	31.8	34.1
4:1	35.5	38.9	25.1	25.9	34.9	37.4	35.0	38.2

Table 1. Comparison between single structure and proposed architectures in terms of PSNR.

The parallel-structure architecture resulted in PSNR = 33.8dB (for CR = 8:1) for “Peppers” when trained with “Lena”, and 33.0dB when trained with “Baboon”. The PSNR for the proposed technique is 35.1dB for the same CR. Moreover, the dependence of the parallel-structure NN on the training data becomes apparent.

An example that presents a visual comparison of the compression results is shown in Figure 9. Figure 9(a) presents a portion of the original image. Figures 9(b), 9(c) and 9(d) present the same portion of the compressed image using, respectively, the proposed algorithm, the single NN architecture and the parallel structure architecture. The compression ratio used was 10:1. Although the blockage effect is apparent in all

compressed images, the proposed algorithm presents the best visual result especially around edges.

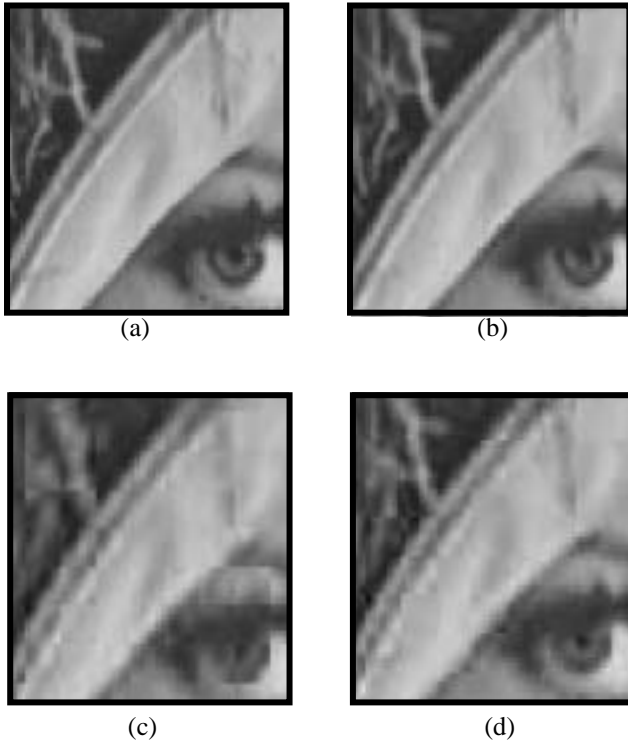


Figure 9. Images: (a) original, (b) compressed with proposed algorithm, (c) compressed with single structure NN, and (d) compressed with parallel structure NN.

4.1.2 Comparisons in terms of computational complexity

Generally, NN techniques exclude training from the encoding process. Network(s) are trained with images other than the ones to be encoded. This subsection compares the parallel-structure and the proposed method in terms of required operations. As mentioned previously, the proposed method includes training in the encoding process.

The total number of hidden nodes in the parallel architecture is 40. In the assignment process, all patterns have to be presented to all four networks. This requires a total of $40 \text{ (nodes)} \times 64 \text{ (pattern size)} \times 4096 \text{ (blocks)} \times 2 \text{ (3 layer NNs)} \approx 21 \cdot 10^6$ multiplications and $40 \times 64 \times 4096 = 10.5 \cdot 10^6$ additions for a 512×512 image. Additionally, there are $3 \times 40 \times 64 \times 4096 = 31.5 \cdot 10^6$ operations required for the calculation of the error patterns. Hence, a total of $63 \cdot 10^6$ operations are required for encoding. This number does not include the iterative optimization algorithm which is also part of encoding.

On the other hand, the number of operations for the

cascade architecture is variable due to the adaptive nature of the technique. The number of iterations per unit is set to 4. It was found that for “Lena” (size 512×512), and for CR of 8:1 the number of multiplications and additions were $13 \cdot 10^6$ and $13 \cdot 10^6$ respectively, for presenting the patterns to the architecture. Furthermore, there were $19.5 \cdot 10^6$ operations required for the error pattern calculation. Hence, a total of $45.5 \cdot 10^6$ operations were required for encoding. Similarly for “Baboon” (size 512×512), a total of $53.2 \cdot 10^6$ operations were required for encoding. Therefore, although the proposed algorithm includes training in the encoding process, it requires fewer operations than the parallel NN technique.

4.2 Comparisons with JPEG

This section presents comparison results between the complete proposed algorithm and JPEG. The comparisons presented between the proposed technique and NN techniques did not consider incorporating any lossless compression component in their process.

4.2.1 Comparisons in terms of PSNR

Figure 10 compares JPEG and the proposed algorithm for the “Baboon” image and for different compression ratios. It can be concluded that the proposed technique provides higher PSNR than JPEG for high compression ratios. The PSNR difference increases with the compression ratio.

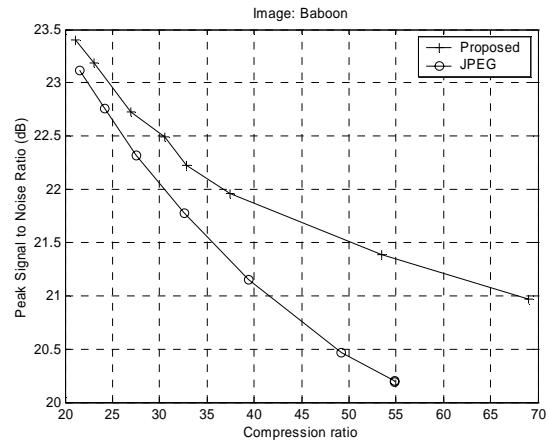


Figure 10. Comparison of proposed algorithm and JPEG for the “Baboon” image

Figure 11 depicts another comparison example. In this example, the two algorithms are compared for the texture image of Figure 8(a).

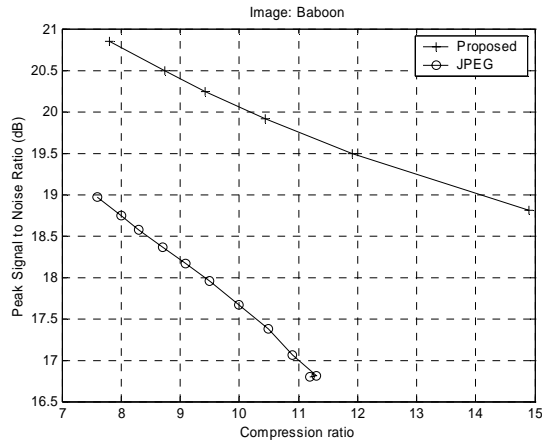


Figure 11. Comparison of proposed algorithm and JPEG for the texture image

In Figure 11, the superiority of the proposed technique is more apparent. The texture image cannot be significantly compressed while maintaining high PSNR. The image is rich in information since it contains both significant high and low frequency components. The adaptive technique produces significantly higher PSNR than the fixed transform based JPEG. From the results presented in Figure 11 it can be observed that the proposed technique is around 2-3 dB better than JPEG. Again, higher compression ratios result in larger PSNR differences between the two techniques.

4.2.2 Comparisons in terms of computational complexity

Since quantization and lossless coding are components of both JPEG and the proposed algorithm, the computational complexity of the two techniques is compared in terms of the associated transforms (DCT vs. adaptive).

The number of multiplications required for the two-dimensional DCT of an 8×8 block using fast implementation of the DCT is 584 [10]. For instance, a 512×512 image (4096 blocks) would require a total of $2.4 \cdot 10^6$ operations.

The proposed technique requires a variable number of operations. An estimate of the number of operations required is provided in this section. Each block requires approximately 64 multiplications and 64 additions for equation (5), a total of 128 operations for a single transformation. It also requires approximately 65 multiplications and 65 additions, a total of 130 operations for equation (6). Moreover, it requires approximately 64 multiplications and 64 additions for the error calculation, a total of 128 operations. Finally, there are 128 operations required for every error calculation of the adaptive quantization step. Overall,

there is a total of approximately $NT \times IT \times (256 + QIT \times 128)$ operations required per block, where NT is the average number of transformations per block, IT is the number of iterations per unit for calculation of coefficients and weights, and QIT the number of iterations required in the adaptive estimation of the coefficient quantization levels. If for instance $NT = 6$, $IT = 4$, and $OIT = 5$, the number of operations is equal to 21504 per block. This is approximately 37 times slower than JPEG's equivalent component. On the other hand, with today's computational power the actual time requirements are reasonable. For instance, a 1.5GHz Pentium 4 computer requires around 5 seconds to code a 512×512 image for a 30:1 compression ratio.

5. CONCLUSIONS

This paper proposes a novel adaptive algorithm for image compression. The major advantage of this algorithm is that it requires a small number of training parameters and has a fast training phase compared to other adaptive techniques such as neural networks (NN). Therefore, training can be incorporated into the encoding process. The proposed architecture exhibits smaller coding error than previous single structure and parallel structure NN architectures. Moreover, although the training phase is included in the encoding process, it exhibits faster encoding than parallel structure NN techniques.

The overall process of the proposed algorithm exhibits similarities to the JPEG algorithm. The major differences are the substitution of DCT with adaptive transforms, and the adaptive estimation of coefficient quantization levels. The proposed technique produces higher PSNR than JPEG, especially for high compression ratios. Although the proposed method is more computationally complex than JPEG the compression time required is reasonable.

REFERENCES

- [1] Cottrell, G.W., Munro, P., Zipser, D. "Image compression by backpropagation: an example of extensional programming," in Sharkey, N.E., (ed.) *Models of cognition: a review of cognition science*, NJ:Norwood, 1989.
- [2] Carrato, S., Marsi, S. "Parallel Structure Based on Neural Networks for Image Compression," *Electronics Letters*, Vol.28, No.12, pp. 1152-1153, June 1992.
- [3] Benbenisti, Y., D. Kornreich, H.B. Mitchell, and P.A. Schaefer, "Fixed Bit- Rate Image Compression Using a Parallel-Structure Multilayer Neural Network,"

- IEEE Trans. on Neural Networks*, Vol.10, No. 5, pp. 1166-1172, Sept. 1999.
- [4] A. Basso and M. Kunt, "Autoassociative neural networks for image compression," *European Trans. Telecommun.*, vol. 3, pp. 593-598, 1992.
- [5] G. Qiu, M. R. Varley, and T. J. Terrell, "Image compression by edge pattern learning using multilayer perceptrons," *Electron. Lett.*, vol. 29, pp. 601-603, 1993.
- [6] H. M. Abbas and M. M. Fahmy, "Neural model for Karhunen-Loeve transform with application to adaptive image compression," *Proc. Inst. Elect. Eng.*, vol. 140, pt. I, pp. 135-143, 1993. See also errata in *Electron. Lett.*, vol. 28, p. 1562, 1992.
- [7] M. E. Blain and T. R. Fischer, "A comparison of vector quantization techniques in transform and subband coding of imagery," *Signal Processing: Image Commun.*, vol. 3, pp. 91-105, 1991.
- [8] C. Cramer, E. Gelenbe, H. Bakircioglu, "Low bit rate video compression with neural networks and temporal subsampling," *Proceedings of the IEEE*, Vol. 84, No. 10, pp. 1529-1543, October 1996.
- [9] Chang, G. G. Langdon Jr., and J. L. Murphy, "Compression gain aspects of JPEG image compression," in *Proc. SPIE Image Processing Algorithms Techniques III*, San Jose, CA, Feb. 10-13, 1992, pp. 159-168.
- [10] G. Bi, G. Li, and K.K. Ma, "On the computation of two dimensional DCT," *IEEE Transactions on Signal Processing*, vol. 48, No. 4, pp. 1171-1183, April 2000.
- [11] G.K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, Vol. 38, No 1, Feb. 1992.
- [12] W.B. Pennebaker and J.L. Mitchell, "JPEG: Still image data compression standard," *Van Nostrand Reinhold*, 1993.
- [13] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image compression coding system: an overview," *IEEE Transactions on Consumer Electronics*, Vol. 46, No 4, Nov. 2000.