1. Introduction

Image compression is essential for applications that demand a high efficiency in storing and transmitting images. Such applications include medical images database, pictorial catalog archives, facsimile transmission, video conferencing, and multimedia systems, just to name a few.

Consider a full-color image of 727×525 pixels with 24 bits per pixel. Without compression, it takes more than 1.1 megabytes of memory to store just one image. Let *compression ratio* be defined as the ratio of the size of the original image versus the compressed image. By using JPEG† at its default setting, a compression ratio of 20 can be achieved [1]. And the corresponding decompressed image is nearly indiscernible from the original. If image quality loss is tolerable, say for the purpose of image previewing, a compression ratio as high as 170 or more is achievable by JPEG. Direct advantages can be realized for storing and transmitting images in such a compact form.

There are many available image compression techniques today, and these techniques are normally evaluated according to their compression ratios, compression/decompression times, and the decompressed image fidelity. To define the decompressed image fidelity, let x_{ij} be a pixel of an image of P×Q pixels and $\bar{x}_{ij}$ be the corresponding pixel of the decompressed image, then the *mean square error* (MSE) is defined by

$$\mathrm{MSE} = \frac{1}{PQ}\sum_{i=1}^{P}\left(\sum_{j=1}^{Q}(x_{ij}-\bar{x}_{ij})^2\right).$$

A widely adopted distortion measure for fidelity, the *signal-to-noise ratio* (SNR), can then be expressed as

$$\mathrm{SNR}\,(\mathrm{in\,dB}) = 10\cdot\log_{10}\frac{(2^n-1)^2}{\mathrm{MSE}},$$

where n is the number of bits per pixel in the image.

Neural networks have been exploited in many applications of image processing and pattern recognition. Among these applications is the recent interest in applying neural networks for image compression. However, researchers tend to solve the same problem through rather different network structures as they view the compression process quite differently. The goal of this report is to survey and classify the existing neural network based methods for image compression.

This report starts with a brief review of artificial neural networks. The next three sections cover various neural network based methods which are classified according to whether they are direct methods, vector quantization methods, or other related methods.

---

† JPEG (Joint Photographic Experts Group) is a standardized image compression scheme based on block discrete cosine transform and scalar quantization.

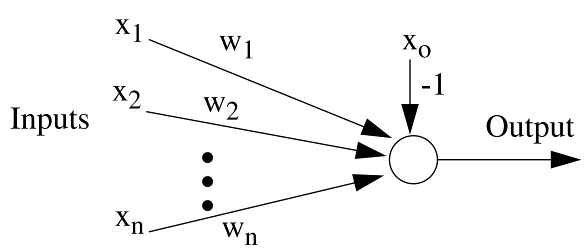
2. A Review of Artificial Neural Networks

Artificial neural networks are machine models of the biological neural systems with the aim of achieving human-like performance for the tasks that are difficult, if possible at all, for conventional computers to cope with and yet are quite easy for us to handle. Pattern recognition is a good example. Though the history of neural network research can be traced back to late last century [2], a recent resurgence of the field that drew much attention from researchers of various disciplines is due to new network topologies (feed-forward multilayer network, Hopfield feedback network), algorithms (back propagation, stochastic neural network), implementation techniques (digital/analog VLSI technology, electro-optics technology), and various emerging applications.

A neural network is an assembly of simple processing elements, neurons, connected according to certain network topology. Figure 1 shows the functionality of a typical neuron. An artificial modeled neuron consists of inputs, input weights, a summation function, a transfer function, and an output. The summation function sums the weighted inputs. Inputs and connection weights can be of binary or analog value. The connection weight in the model corresponds to the correlating strength between the *presynaptic potential* and the *postsynaptic potential* of a real neuron. A positive connection weight implies a positive correlation between the connected neurons or a rewarding relationship, and while a negative weight implies negative correlation or a punishing relationship.

The outcome of the summation function is passed to the transfer function to produce the output. Four transfer functions typically used are shown in Figure 2 and they are *step*, *hard-limit*, *linear*, and *sigmoid* functions. The sigmoid function is most pervasive because it is bounded and nondecreasing and provides a graded, nonlinear response which most resembles a real neuron. The constant *a* in the sigmoid function is called the *neuron gain*. It controls the nonlinearity of the sigmoid function. Note that for step, hard-limit, and sigmoid functions, the range of the output need not to be [0, 1]. Depending on the applications, the range can be rescaled to [-1, 1], giving symmetry with respect to the origin.

To perform a desired task, neurons need to be connected to form a network according to a certain topology. There are several ways to differentiate network connection topology: by number of layers, by connections within a layer, and by connection between layers. Figure 3(a) shows a single-layer feedback network. Each neuron has an external bias (input) together with weighted signals from outputs of all neurons. One example of such network is the well-known Hopfield



$$y = f\left(\sum_i w_i x_i - x_o\right)$$

Figure 1. The basic processing element (neuron) of a neural network.

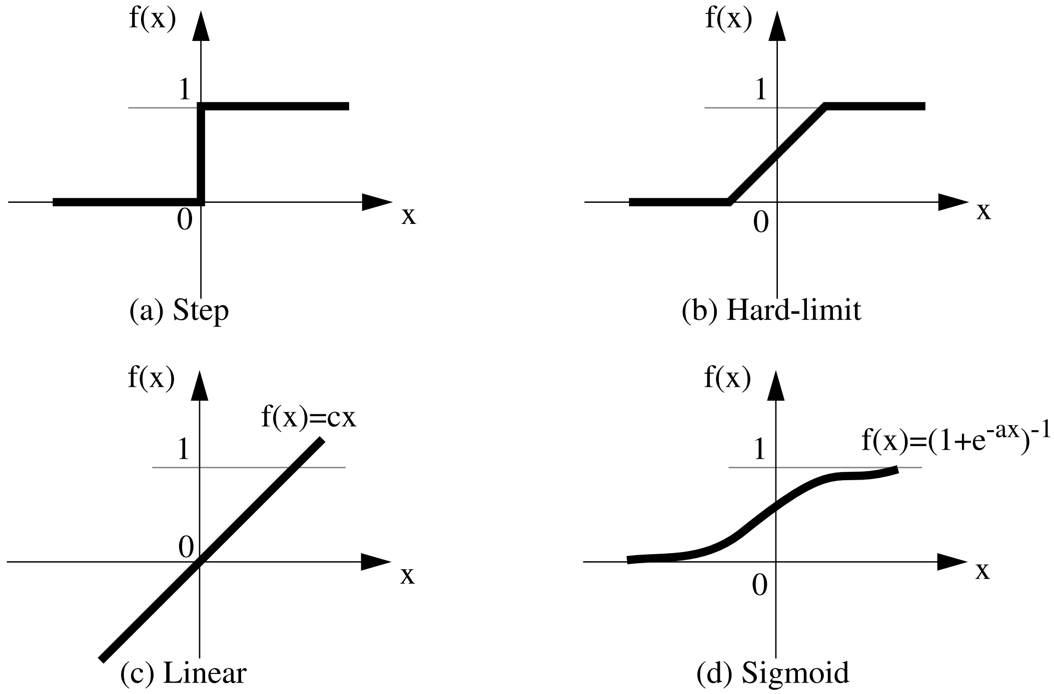


Figure 2. Typical transfer function for a neuron.

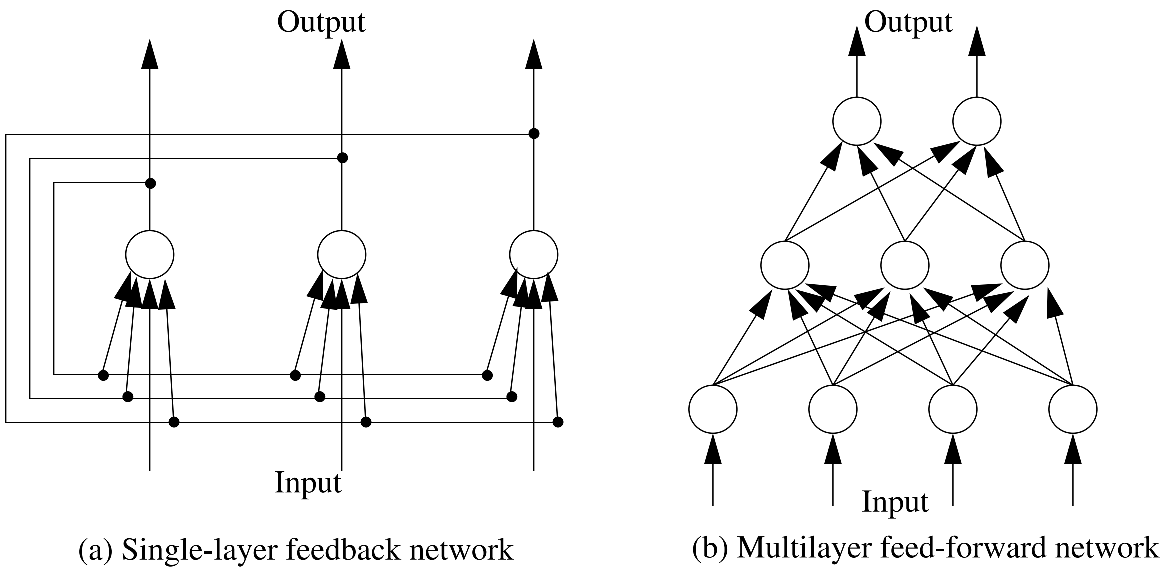


Figure 3. Typical neural network typology.

network for which each neuron has a sigmoid transfer function. Figure 3(b) is a multilayer feed-forward network. The normal direction of information flow is from the input layer toward the output layer. Neurons of the same layer always have the same type of transfer function, but do not necessarily have the same parameter setup for these functions. Though not shown in the figure, every connection is weighted.

Multilayer networks can have bidirectional (feed-forward and feedback) connections between layers and also lateral connections within layers. Neurons within a layer need not to be fully connected, like the one in Figure 3(a). These lateral connections may be nearest-neighbor type connections, or be randomly connected with fixed fan-in and fan-out, or follow certain topological pattern such as mesh. Kohonen's *feature map* is an example of the nearest-neighbor connected network. A complex network can be built with a combination of various kinds of subnetworks. For instance, the Hamming net in [3] is formed by a two-layer feed-forward net cascaded by a single-layer feedback network, where the feed-forward net generates the matching scores of the input vector against all codewords and the feedback net selects the best-match codeword.

A network with a proper connecting topology and suitable transfer function for each neuron may still not be enough to achieve any desired result, since the weight of each connection has not been determined yet. Connection weights between neurons must be properly made. But the problem is that for networks of reasonable size their connection weights cannot in general be preselected. An adaptation process must be done to the network, either on the fly or during a training stage, to tune the connection weights in order for a network to exhibit the desired functionality or intelligence. Such a process is called *learning* in the field of neural networks, indicating that it may exhibit certain behaviors found in natural neural systems.

There are many learning schemes, and they can be divided into two categories, supervised learning and unsupervised learning. With supervised learning, both a test input and the correct output are given to a network. The difference of the network-produced output and the correct output is fed back to the network to adjust connection weights in a way that the difference will be gradually reduced to a pre-selected tolerable range. With unsupervised learning, no correct output is given and the network learns from examples through a pre-set criterion. Figure 4 is a list of networks with different learning methods. These networks are also divided according to whether the network topology is feed-forward or feedback. As new learning methods or variants of existing methods are invented so frequently, the list is by no means complete. The selection of a learning method is made according to the network topology and the neuron transfer function, and sometimes the training time and the performance as well. In this report only two learning methods are described in details because of their often citation in the literature of neural networks for image compression.

Consider a two-layer feed-forward network. Let x_i be the activation (input) of the *i*th neuron in the input layer, y_j and $\bar{y}_j$ be the actual output and the desired output the of the *j*th neuron in the output layer, respectively, and w_{ij} be the corresponding connection weight of these two neurons. Then the so-called *delta rule*, also known as the Widrow-Hoff rule or least mean square (LMS) rule, can be expressed as

$$\Delta w_{ij} = \alpha \cdot (\bar{y}_j - y_j) \cdot x_i,$$

where Δw_{ij} is the adaptation of the connection weight and α a pre-selected learning constant. The connection weight is initialized by a small random value. The delta rule is a supervised learning method as the desired output is given and it updates the weight according to the difference of the desired output and the actual output, and the input excitation. It is called the least mean square

	UNSUPERVIZED LEARNING	SUPERVISED LEARNING
FEEDBACK NETS	Additive Grossberg Shunting Grossberg Binary adaptive resonant theory Analog adaptive resonant theory Discrete Hopfield Continuous Hopfield Discrete bidirectional associative memory Temporal associative memory Adaptive bidirectional associative memory	Brain-state-in-a-box Fuzzy cognitive map Boltzmann machine Mean field learning
FEEDFORWARD NETS	Learning matrix Driver-reinforcement learning Linear associative memory Optimal linear associative memory Sparse distributed associative memory Fuzzy associative memory Learning vector quantization Counterpropagation Competitive learning	Perceptron Adaline & Madaline Backpropagation Cauchy machine Adaptive heuristic critic Time delay neural network Associative reward network Avalanche matched filter Backpercolation Artmap

Figure 4. Neural networks with different learning methods. Networks are classified according to network type, feedback or feed-forward, and learning type, supervised or unsupervised. (Information taken from “FAQ in comp.ai.neural-nets -- monthly posting.”)

rule because it can be interpreted as a gradient descent method which seeks to minimize the mean square error between the desired output and the actual output. The delta rule was developed by Widrow and Hoff in 1960 and applied to a network named Adaline in which each neuron had a linear transfer function.

The backpropagation network is a multilayer feed-forward network adopting what is called the *generalized delta rule* (a generalization of the delta rule to networks with three or more layers) for learning. Each neuron of a backpropagation net has a sigmoid transfer function. For description purpose, consider a three-layer feed-forward network. Let x_i be the activation of the *i*th neuron in the input layer, h_k be the value of the *k*th neuron in the hidden layer, and y_j and $\overline{y}_j$ be

the actual output and the desired output of the *j*th neuron in the output layer, respectively. The sizes of the input, the hidden, and the output layers are *N*, *M*, and *L*, respectively. Connection weights between the input and the hidden layers are denoted by w_{ik}, and weights between the hidden and the output layers by v_{kj}. The backpropagation algorithm for such a network is given in the Box 1. The input vector is fed into the network in a forward fashion to produce the actual output, and then the error of the actual output and the desired output is propagated in a backward fashion, from the output layer to the hidden layer and finally the input layer, to update connection weights (and thus comes the name of backpropagation). Sometimes a faster convergence can be achieved by adding extra terms to weight updating formulas as

$$\Delta v_{kj}(t+1) = \alpha \cdot \delta_j \cdot h_k + \beta \cdot \Delta v_{kj}(t),$$

$$\Delta w_{ik}(t+1) = \alpha \cdot \varepsilon_k \cdot x_i + \beta \cdot \Delta w_{ik}(t)$$

Box 1: The Backpropagation Algorithm

Step 1. Initialize all weights of w_{ik} and v_{kj} to small random values (between [-0.1, 0.1]).

Step 2. Present input vector $x = [x_1, \ldots, x_N]$ to the input layer and propagate the activations from the input layer, to the hidden layer, and then the output layer to produce the actual output $y = [y_1, \ldots, y_L]$.

Step 3. Compute the errors of the neurons in the output layer by

$$\delta_j = y_j(1 - y_j)(\bar{y}_j - y_j).$$

Step 4. Update v_{kj} according to

$$\Delta v_{kj} = \alpha \cdot \delta_j \cdot h_k,$$

where α is a learning constant.

Step 5. Compute the errors of the neurons in the hidden layer by

$$\varepsilon_k = h_k(1 - h_k)\Sigma_j(\delta_j \cdot v_{kj}).$$

Step 6. Update w_{ik} according to

$$\Delta w_{ik} = \alpha \cdot \varepsilon_k \cdot x_i.$$

Step 7. Repeat Steps 2 to 6 for another pair of an input vector and its desired output, till all such training pairs being processed.

Step 8. Repeat Steps 2 to 7 for all training pairs till the connection weights converged.

where $0<\beta<1$, and $\Delta v_{kj}(t)$ and $\Delta w_{ik}(t)$ are updates from last iteration. The backpropagation algorithm can be easily generalized to networks with more than one hidden layer. (For introductory articles of artificial neural networks refer to [3-5] and for further information to [6-8].)

3. Direct Methods

Different neural network based methods have been exploited for image compression. In this section we will discuss the *direct methods*. Two common features of these methods are that they all operate in the image domain, i.e., compressing blocks of pixels directly, and they adopt multilayer feed-forward networks with the same number of neurons at the input and the output layers, and less neurons in the hidden layers.

The first direct method was proposed by Cottrell, Munro, and Zipser in 1987 [9]. The network they used is basically an $n \times m \times n$ three-layer feed-forward network, for $n>m$, with a uniformly spaced scalar quantizer added at the output of each neuron in the hidden layer, as shown in Figure 5. A typical choice of the network size is 64×16×64 with 5-bit quantization (32 quantization levels). The first half of the network, from input layer to the hidden layer plus the scalar quantizers, is regarded as the encoder, and while the second half as the decoder. Each neuron in the hidden layer and the output layer has a sigmoid transfer function with a range of [-1, 1].

The network was proposed to compress a single image. To operate, an image (with 8 bits/pixel) to be compressed is split into non-overlapping patches (subimages) of size n pixels, i.e., n-dimensional vectors. Random samples of subimages are given to both the input and the output ends of the network to update the randomly initialized weights via backpropagation such that the network will reproduce at the output layer (approximately) the same subimage fed in at the input layer. For each subimage fed into the network, the gray levels of its pixels is converted linearly from [0, 255] to [0, 1]. This conversion can be done either before or in the input layer. An oppo-

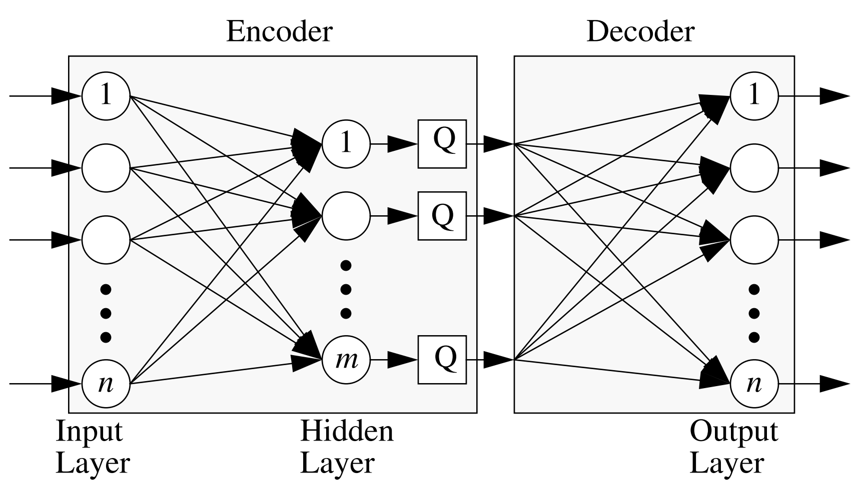


Figure 5. A three-layer feed-forward network used by most direct NN methods.

site conversion is necessary at the output end of the decoder to map outputs with a continuous range [-1, 1] to a discrete range [0, 255]. After a training process, the compressed file is a collection of compressed, quantized subimages. The compression is achieved due to two factors, first, the number of neurons in the hidden layer is less than that of the input layer, and second, the scalar quantization requires less bits (e.g., 5 bits) as compared to the input data (8 bits). With a setup of 64×16×64 network and 5-bit scalar quantization, the corresponding compression ratio is 6.4. And it will be 8.0 if a 4-bit quantization is used instead. Ways to improve the compression ratio include reducing scalar quantization levels, enlarging input subimages, and reducing neurons in the hidden layer, but all come at the cost of SNR degradation.

Sonehara, *et al.*, in [10] used a similar 64×m×64 network for image compression, for m=9, 16, or 25. The differences of their network from the previous one are that the sigmoid transfer function of each neuron has a range of [0, 1], rather than [-1, 1], and that the resolution of the scalar quantization is fixed to 8 bits, same as the input data with 8 bits/pixel. The image to be compressed is also split into proper subimages and then they are fed simultaneously into many replicates of the 64×m×64 net implemented on a NCUBE machine. Different subimages are learned by different subnetworks and each subnetwork has different connection weights. Backpropagation was used for learning. The gray level of each input pixel is converted linearly from [0, 255] to [0, 1], and the final output value from [0, 1] to a discrete range [0, 255]. Sonehara, *et al.*, have used their network to compress multiple learned images as well as multiple unlearned images. The compression ratio is just the ratio of the number of input neurons versus the number of hidden neurons. The SNR has been shown to be affect by several factors: whether an image is learned or not, the number of training iterations, the number of learned images, and the number of hidden neurons. For a 64×16×64 network setup with 19 learned images, each trained with 2000 iterations, the average SNR of learned images is 27.12 dB and for unlearned images 14.22 dB. Table 1 shows how various factors affect the SNR. In all cases, the average SNR of unlearned images is at least 10 dB less than that of learned images. As expected, more hidden neurons results in a higher SNR. When the number of learned images increases, the network learns to compress images of more general forms, and thus the SNR of unlearned images becomes higher. But the SNR of learned images degrades slightly in order to accommodate such a generalization. As the number of training iterations increases, the network can be thought of as too closely tuned with learned images, and thus express less generalization property toward unlearned images. Both random values and discrete cosine transform coefficients have been used for the initialization of connection weights of the network, yet the resultant SNRs indicate no difference.

Images	More hidden neurons	More learned images	More training iterations
Learned	↗	↘	↗
Unlearned	↗	↗	↘

↗ = SNR increases; ↘ = SNR decreases.

Table 1. Factors affect the SNR.

Kohno, Arai, and Imai used 64×8×64 and 64×4×64 networks to compress still images [11]. Rather than using sigmoid transfer function for each neuron, a partial-linear and partial-nonlinear transfer function was used. The function is linear in [-*c*, *c*] and otherwise sigmoid, for *c* a predetermined constant. The gray level of each input pixel is converted linearly from [0, 255] to [0, 0.8]. The main feature of their network is that both connection weights and the neuron gain (nonlinearity factor) of each neuron are trained during learning process. This is due to the argument that updating of both weights and neuron gains can achieve better learning performance than updating weights alone. A particular learning algorithm, based on backpropagation, was used to update weights and neuron gains. Each neuron when using such a learning scheme could end up with a different neuron gain. Simulation results substantiated the argument as the SNR of the compressed image thus obtained was higher. But the tradeoff is that the network takes a much longer training time. Furthermore, since the network is specially trained for a single image and thus the network parameters are image-dependent, both the weights from the hidden layer to the output layer and the neuron gain of each neuron in the output layer, plus the quantized output from the hidden layer, need to be transmitted to the decoder for decompressing a particular image. This yields a few percent reduction to the compression ratio.

For multiple images compression, Anthony, *et al.*, compared the results of a neural network approach and a conventional method, namely, the principal component analysis (PCA) [12]. The networks they used were the same as of Cottrell, *et al.*, with size 64×16×64 and 512×128×512. For the sole purpose of comparing the compressed image fidelity, (not caring for the compression ratio) no scalar quantization is taken after the hidden layers of the networks. It is found that for learned images, the network produced comparable results to PCA, but much worse for unlearned images. This situation is much more evident for the larger network (512×128×512) than for the smaller one, though they yield same compression ratio. (For more on PCA, refer to [7].)

Namphol, Arozullah, and Chin developed a five-layer feed-forward network for image compression [13]. The network was of size 512×64×*m*×64×512 and was trained according to the so-called *nested training rule*. In the beginning, a three-layer network of size 64×8×64 called the *outer net*, same as of Cottrell *et al.*, is trained with all subimages of size 8×8 pixels by backpropagation in order to reproduce the input subimage at the output layer. Then a 64×*m*×64 network called the *inner net* is used. The input to the inner net is a collection of the output of the hidden layer in the outer net with respect to four different subimages. The inner net is trained again by backpropagation to reproduce its input at its output end. After training period, a network of size 512×64×*m*×64×512 is built with its three hidden layers identical to the inner net just trained. Its connections from the input layer to the first hidden layer are eight replicates of the connections of the first half (from the input layer to the hidden layer) of the outer net. Similarly, the connections from the last hidden layer to the output layer are eight replicates of the second half of the outer net. Note that the connections between three hidden layers are fully connected, but the connections to the input and the output layers are only blockly connected. The reason for training the five-layer network in such a cascade fashion, rather than training it as a whole, is that the training time can be considerably reduced. As a network with more hidden layers is supposed to bear in its connection weights more interrelation among subimages, this five-layer net is expected to achieve a higher compression ratio at an acceptable SNR. The simulation results vindicate this conjecture. Various values of *m* for the network have been tried and the SNR for a trained image is a few dB

higher (better) than that of an untrained image. With m=32 and 8-bit scalar quantization after the hidden layer of the inner net, equivalent to a compression ratio of 16, an SNR of 22.68 dB was obtained for a trained image, and 16.35 dB for an untrained image. If 5-bit quantization is used instead, equivalent to a compression ratio of 25.60, then the SNR for the trained image becomes 19.73 dB, which is still acceptable for many uses.

Mougeot, Azencott, and Angeniol treated a linear three-layer feed-forward network as an auto-association memory and used it to compress single image [14]. The network is usually of size 64×8×64 and each neuron has a linear transfer function. To apply the network, an image is split to 8×8 subimages and each subimage has an overlap of 1 pixel with its neighbor subimages. This subimage overlapping aims at reducing the blocking artifact when nonoverlapping subimages are used. Experiments showed that with subimage overlapping the network achieves a slightly higher SNR value at the cost of a lower compression ratio. The values of all subimages' pixels at any particular pixel position are transformed in a way that input pixels to the corresponding neuron in the input layer have a mean 0 and a variance 1. After such an input transformation, subimages are fed to the network one by one and a linear version of backpropagation is used for learning. The authors have exploited several possible features that would increase the network performance. It was found that using 8×8 subimages results in less normalized mean square error than using 16×16 subimages. As the neuron transfer function is linear, the network can be regarded as a two-phase cascaded linear mapping, one phase from the input layer to the hidden layer and the other phase from the hidden layer to the output layer. The mapping of the first phase is a 64×8 matrix and the second phase a 8×64 matrix. Since there are infinitely many pairs of matrices that would achieve the same goal of reproducing best-match decompressed images at the output end of the network, the authors chose to preset 8×8 = 64 positions of the first matrix as an identity matrix. This leads to a faster weights convergence during linear training process. Also for constructing the error measure function, L_1 norm was found to be more suitable than L_2 norm (the often-used quadratic error measure) for images with larger homogeneous portion, and thus achieved better visual appearance for the decompressed image. A similar result has been confirmed by other researchers [15]. Lastly, since the network is linearly trained and thus the weights are linearly dependent on the subimages used for training, weights are image-dependent. Every image leads to a different set of connection weights. This would impair the compression ratio by a few percent as the compressed data now includes the quantized output of the hidden neurons and the weights from the hidden layer to the output layer.

4. Vector Quantization Methods

Vector quantization has been extensively studied during early 1980's for the purpose of data compression [16]. An excellent survey paper on using vector quantization for image coding was written by Nasrabadi and King [17]. Different from the scalar quantization used in direct compression methods just described, vector quantization maps a sequence of vectors, or subimages in image compression context, to a sequence of indices according to a codebook of reference vectors, and the sequence of indices which is much more compact than the original vectors is stored or transmitted through a transmission channel. With the same codebook at the decoder end,

the sequence of the indices can be readily mapped to the corresponding reference vectors in the codebook. A block diagram of vector quantization is given in Figure 6. A codebook of, say, k reference vectors is generated during a training stage. When an input vector x is presented, the vector quantization encoder selects the best-match or the closest (in terms of Euclidean distance) vector m_i from the codebook. Then the encoder outputs the index of the best-match vector for transmission. The codebook needs only to be generated once and after that both the encoder and the decoder will have a copy of it. The decoded vector $\bar{x}$ is the best-match (quantized) vector m_i for the input vector x. As the encoding and the decoding processes are straightforward, it is clear that to have a good vector quantizer the bottom-line is to have a good codebook, or a good scheme to generate and, if desirable, update the codebook.

Given a fixed-size memory for codebook storage, the goal of codebook generation, that is the goal of vector quantization, is to find for all input vectors a set of reference vectors that would minimize the mean square error between the input vectors and their best-match reference vectors. If all input vectors or the statistical distribution of input vectors are available, an optimal codebook can be generated via proper statistical analysis. But generally, this is not the case. Often a codebook is designed with only a limit number of test vectors available and with no *a priori* knowledge of the input vector distribution. A typical procedure for codebook generation is outlined in Figure 6. The codebook is initialized either randomly or according to certain heuristics, for example, K-means clustering. Then the codebook adapts its content by certain training scheme. In this section we examine in details the neural-network-related vector quantization

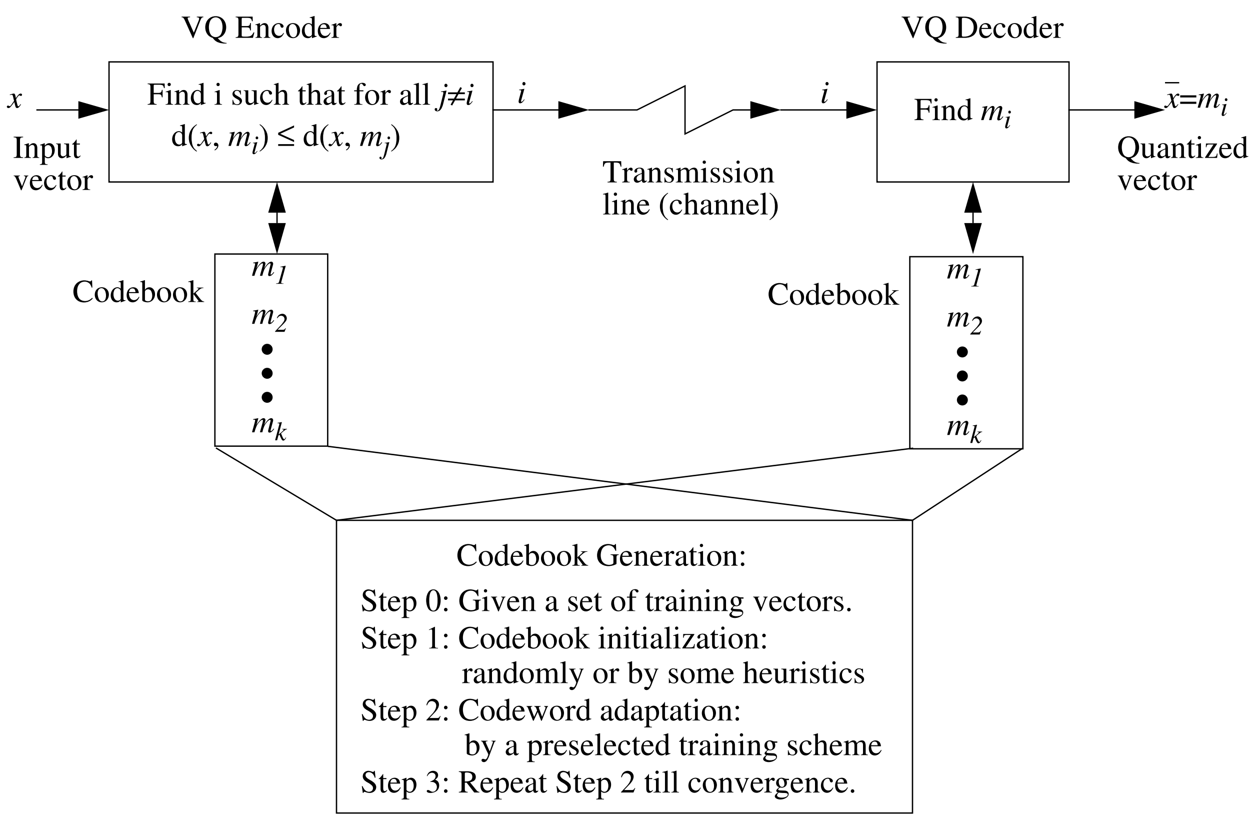


Figure 6. A block diagram of vector quantization.

methods for image compression. More information on conventional vector quantization methods can be found in [16,17].

Let $x(t)$ be the training vector and $m_i(t)$, for $1 \le i \le k$, be the reference vectors at step t during training. And each vector is an n-dimensional vector, or a subimage of n pixels in the context of image processing. In the sequel the initial reference vectors $m_i(0)$, for $1 \le i \le k$, are assumed to be either randomly chosen or be the first k training vectors from a training set, unless otherwise mentioned.

The so-called *learning vector quantization* (LVQ) coined by Kohonen actually has two different versions: supervised and unsupervised [18]. The unsupervised LVQ is also known as the *competitive learning* algorithm. The network for unsupervised LVQ can be realized as a three-layer network with n input neurons, k hidden neurons, and k output neurons as shown in Figure 7. During training step t, each input neuron is presented the corresponding component of $x(t)$ and it passes this component to the hidden layer. Each neuron in the hidden layer has a local memory of the corresponding $m_i(t)$, and it calculates the distance of $m_i(t)$ to $x(t)$. The results of hidden neurons are forwarded to the output layer. The output neurons compete with each other such that the neuron with its $m_i(t)$ closest to $x(t)$ emits an output $y_i = 1$, and other neurons emit 0's. The reference vector updates according to the following rule,

$$m_i(t+1) = m_i(t) + \alpha(t) \bullet [x(t) - m_i(t)] \bullet y_i, \text{ for } 1 \le i \le k,$$

where $0 < \alpha(t) \le 1$ is a learning factor that decreases monotonically during learning. The update formula basically moves the winning $m_i(t)$ toward the training data $x(t)$ and leaves the rest of the reference vectors unchanged.

For supervised LVQ, all vectors come from a certain number of classes and the classification of each vector (training or reference) is known beforehand. Let the training vector $x(t)$ belong to class $C(x)$ and the reference vector m_i class $C(m_i)$. Several reference vectors may belong to one class. Let $I(i)$ be an indicator such that $I(i) = 1$ if $C(x) = C(m_i)$, and $I(i) = -1$ if $C(x) \ne C(m_i)$. Then the supervised LVQ network includes a network same as the unsupervised LVQ net, plus a separate subnetwork that computes the score of $I(i)$, for $1 \le i \le k$. The reference vector then updates by

$$m_i(t+1) = m_i(t) + I(i) \bullet \alpha(t) \bullet [x(t) - m_i(t)] \bullet y_i, \text{ for } 1 \le i \le k.$$

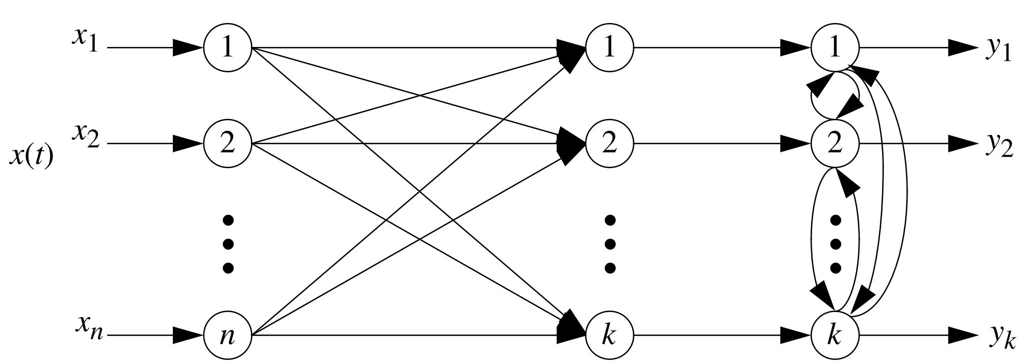


Figure 7. The Kohonen unsupervised LVQ network.

If the winning $m_i(t)$ and the training vector $x(t)$ belong to the same class, the winning $m_i(t)$ is rewarded by moving a bit closer to $x(t)$. If, however, they belong to different classes, the winning $m_i(t)$ is punished by moving away from $x(t)$. The rest of the reference vectors remain unaltered. Though the adding of $I(i)$ to the update formula has the effect of minimizing the number of misclassifications, it also moves reference vectors away from the class borders and thus creates depletion zones in these borders [18].

The Kohonen *self-organizing feature map* has also been used for vector quantization, though it was originally used mainly for pattern classification, rather than image coding [18]. It is similar to unsupervised LVQ, but differs in that neurons in the output layer form a lattice and each output neuron takes a position in the lattice. Let $O(t)$ be a topological neighborhood defined for each output neuron at step t. $O(t)$ has a property that it shrinks from step to step. Assume that at step t the nearest reference vector to a given input $x(t)$ is $m_j(t)$, then for an output neuron i a neighborhood indicator $N_i(j)$ (for j is the index of the nearest reference vector) is defined to be 1 if neuron i is within $O(t)$ neighborhood of neuron j in the lattice (the output layer), and to be 0 if otherwise. Reference vectors are now updated by

$$m_i(t+1) = m_i(t) + \alpha(t) \bullet [x(t) - m_i(t)] \bullet N_i(j), \text{ for } 1 \leq i \leq k.$$

Thus the feature map not only updates the winning vector $m_j(t)$, but also those reference vectors that their corresponding neurons in the lattice are within $O(t)$ neighborhood of neuron j, for j the index of the nearest reference vector. As t gets larger, $O(t)$ becomes smaller, and eventually it is so small and contains no other neurons except neuron i itself, for all i, in which case the feature map behaves essentially the same as an unsupervised LVQ. Note that the output layer (lattice) of the feature map can also be replaced by a q-out-of-k network ($q<k$), for which q out of k reference vectors closest to $x(t)$ are chosen to be updated. Depending on whether the classification information is available or not, these q reference vectors update through either supervised or unsupervised LVQ formulas.

The asymptotic nature of reference vectors trained by self-organizing feature map or LVQ tend to approximate the probabilistic distribution of input vectors. But with a limited number of training vectors as input, a certain choice of network parameters greatly affects the rate of a network approaching to its asymptotic behavior. And this in turn determines the goodness of a image compression scheme thus developed. Nasrabadi and Feng in [19] adopted a self-organizing feature map for image compression and chose

$$\alpha(t) = c_1 e^{\left(-\frac{t}{T_1}\right)} \quad \text{and} \quad O(t) = c_2 + c_3 e^{\left(-\frac{t}{T_2}\right)},$$

where c_1, c_2, c_3, T_1, and T_2 are constant parameters. Two sets of parameters were used, depending on whether the subimage to be compressed is edge-like or homogeneous. A codebook contains 348 4-dimensional reference vectors was formed and 284 of these vectors were edge patterns and the rest homogeneous patterns. For a test image, a compression ratio of 2.72 at SNR = 30 dB has been reported. The results was also shown to be comparable with the well-known LBG (Linda-

Buzz-Gray) algorithm in terms of compression ratios and SNRs.

Naillon and Theeten used a spin-glass-like network, similar to discrete Hopfield network, to design vector quantization codebook for still images [20]. In their network, 512 neurons were used to compress 2 pixels (256 neurons for 1 pixel as there are 256 gray levels per pixel). Thus each vector is of 512 components and each of which is +1 or -1. The codebook is initialized with a fixed number of uniformly distributed reference vectors. Then during a training process a number of so-called induced vectors are generated via a *minimal overlapping rule*. The codebook now containing both the original reference vectors and induced vectors is adapted through the network self-organization. But it is not clear how the network self-organizing itself. A compression ratio of 3.2 has been reported for a codebook of size 50. However, no SNR information was given. It is suspected that since the results showed no benefit of dealing with a vector space of 512 dimensions, a much smaller network that deals with a vector space with much less dimensions could achieve the same level of compression performance.

Vector quantization by either LVQ or the self-organizing feature map has the problem that due to improper codebook initialization a subset of reference vectors might be consistently chosen and updated, and this causes under-utilization of the rest of the reference vectors. In dealing with such a problem, Ahalt, *et al.*, proposed a frequency-sensitive competitive learning neural network which, in updating reference vectors, takes also into account the number of times a reference being updated [21,22]. Their network is a three-layer network similar to the one in Figure 7, except that to each reference vector a frequency counter μ_i is associated and a frequency function $\Phi(\mu_i)$ is computed. With a given training vector $x(t)$, each neuron in the hidden layer calculates $\|x(t) - m_i(t)\|$, the distance between $m_i(t)$ and $x(t)$, and then the product of $\Phi(\mu_i)\bullet\|x(t) - m_i(t)\|$ is sent to the corresponding neuron in the output layer. Output neurons compete with each other such that the neuron with the smallest product value will emit $y_i = 1$, and the rest $y_i = 0$. The reference vector is updated by

$$m_i(t+1) = m_i(t) + \alpha(t) \bullet [x(t) - m_i(t)] \bullet y_i, \text{ for } 1 \le i \le k,$$

where

$$\alpha(t) = ce^{\left(-\frac{\mu_i}{T}\right)}$$

with c and T being constants. Whenever a reference vector is chosen ($y_i = 1$) for updating during training, its frequency counter increases by 1. In effect, $\Phi(\mu_i)$ serves as a penalty factor that monitors the fairness of the selection and update of reference vectors such that sometimes the least used reference vector will be chosen even though it is not the nearest reference vector. Several $\Phi(\mu_i)$'s have been performed and it was found to be desirable that $\Phi(\mu_i)$ should be more frequency-sensitive in the early stage of the training process and become less sensitive later on. Experiments were done with 4-dimensional vectors and various choices of codebook size. The largest codebook was of size 64. The network-compressed images produce comparable SNRs to that of LBG algorithm. More interestingly, the reference vectors generated by the network were found similar to those by LBG algorithm.

Baras and LaVigna showed in the case of supervised LVQ, if the initial classifications of reference vectors contradict the majority classifications of training vectors, this might lead to reference vectors divergence [23]. A way to remedy such a drawback is to reclassify every reference vector after each training iteration according to the majority vote of training vectors closest to that reference vector, though there was no experimental results given.

Kohonen in [24] proposed a variant of supervised LVQ in which two vectors were updated at each training step. Let $m_i(t)$ and $m_j(t)$ be the two nearest reference vectors to a given training vector $x(t)$. If $x(t)$ and $m_i(t)$ belong to the same class and $m_j(t)$ another class, and if $x(t)$ lies in a zone area defined in-between $m_i(t)$ and $m_j(t)$, then these two reference vectors are updated by

$$m_i(t+1) = m_i(t) + \alpha(t) \bullet [x(t) - m_i(t)],$$
$$m_j(t+1) = m_j(t) - \alpha(t) \bullet [x(t) - m_j(t)],$$

and the rest of the reference vectors remain unchanged. It does not matter whether $m_i(t)$ or $m_j(t)$ is the nearest vector, so long as the assumption is satisfied. For if the assumption is not satisfied, the original supervised LVQ is used instead. The effect of this two-vector adaptation is to shift the mid-plane of two nearest reference vectors to the place where the Bayes border is supposed to lie. Experiments have been done with speech signal and the results showed that the two-vector adaptation LVQ is better in terms of SNRs. Inclusion of the third nearest reference vector for adaptation has also been tried, and demonstrated a positive effect on SNRs.

In [25] the unsupervised LVQ network was modified to include one frequency counter μ_i for each neuron in the hidden layer. As the final output y_i of the network is generated the same as that of the unsupervised LVQ, reference vectors update themselves according to

$$m_i(t+1) = m_i(t) + \alpha(t) \bullet [x(t) - m_i(t)] \bullet y_i, \text{ for } 1 \le i \le k,$$

where $\alpha(t) = 1/\mu_i$ if $\mu_i \le c$ and 0 if $\mu_i > c$, for c a pre-selected threshold value. During training, when a reference vector is chosen ($y_i = 1$) for updating, its frequency counter increases by 1. For this network, the learning rate of reference vectors is frequency sensitive, but not the selection of a reference vector, while for the network in [21] both the learning rate and the reference vector selection are frequency sensitive. Though the network achieves a slightly inferior SNR compared with LBG due to its simple updating rule, it has been implemented by highly parallel VLSI architecture and resulted in a speedup of 750 times over SUN-3/60 implementation. Experiments were done with subimages of size 5×5 and with codebook sizes 64 and 1024, corresponding to compression ratios of 33 and 20, respectively.

Lee and Peterson proposed an adaptive network structure for vector quantization [26]. In their network, not only the connection weights are adjustable, but also the number of neurons which corresponds to the number of the reference vectors. As far as the adjustment of connection weights is concerned, the basic structure of their network is similar to the self-organizing feature map. But extra measures are calculated in order to determine when to add or delete neurons. When the on-line computed error contribution of a neuron is larger than a preset threshold, implying that the input vector distribution is under-represented by the neuron, an extra neuron is added in order to better reflect the input vector space. A neuron will be deleted either if it has not been

active for some time or if its corresponding reference vector is very similar to that of other neurons, implying that the input vector distribution is over-represented. Ways of adding or deleting neurons from the lattice (the final layer) of the self-organizing feature map have been given and simulation results are comparable to the LBG algorithm. By taking advantage of the fact that consecutive input vectors with high correlation will be quantized to adjacent reference vectors, an lattice-index-residual VQ coding scheme has also been proposed and yields a better compression to the LBG algorithm for highly correlated input vectors. The drawbacks of this network structure are that it is computation-intensive and the residual VQ coding performs worse for uncorrelated input vectors.

Yair, Zeger, and Gersho showed the reduction of the distortion function of the self-organizing feature map was not necessarily monotonic, and by incorporating the principles of stochastic scheme into the feature map, they proposed a “soft” competitive learning scheme which would remedy the non-monotonic problem [27]. Their network can be implemented with a two-layer feed-forward network with the number of input neurons equal to vector dimension and the number of output neurons the numbers of reference vectors. It is similar to the network in Figure 7 without the last layer, and yet the functionality of the output neuron is much more complicated. When an input vector $x(t)$ is given, the probability that output neuron i ($1 \le i \le k$) will be winning is calculated by

$$P_i(t) = \frac{e^{-\beta(t)\,\|x(t) - m_i(t)\|^2}}{\sum_{j=1}^{k} e^{-\beta(t)\,\|x(t) - m_j(t)\|^2}},$$

where $\beta(t) = \beta_o c^{t/k}$, with β_o and c being constants and k the number of reference vectors. A probability counter $\mu_i(t)$ is associated with each output neuron and it is updated by

$$\mu_i(t) = \mu_i(t\text{-}1) + P_i(t).$$

Note that $\mu_i(t)$ plays a similar role as the frequency counter in some of the networks mentioned earlier. Now every reference vector gets updated according to

$$m_i(t+1) = m_i(t) + \alpha(t) \bullet P_i(t) \bullet [x(t) - m_i(t)],$$

where $\alpha(t) = 1/\mu_i(t)$. What differs from the competitive learning is that in this case every reference vector is updated toward $x(t)$ proportionally to its winning probability, rather than only one winning vector gets updated. The choice of $P_i(t)$ has the effect that when $\beta(t)$ is small, all reference vectors migrate toward $x(t)$. And as t becomes larger, $\beta(t)$ becomes larger, which causes $P_i(t)$ peaking around the nearest reference vector and thus separates reference vectors from each other. As $t \to \infty$, this scheme turns into competitive learning, for only the nearest reference vector is updated. For the purpose of speeding up the convergence of the algorithm, each $\mu_i(t)$ is reinitialized to 1 after a time frame of training, and the time frame is chosen such that its length gets longer and longer as time progresses. The increase of the frame length allows more updates being done within a time frame when $\beta(t)$ has a larger value. This has the effect of increasing the probability that the final reference vectors are globally optimally chosen. For experiments on speech

signal, the soft competitive learning yields better SNRs as compared to the LBG algorithm. A convergence analysis of the self-organizing feature map is also provided in [27].

5. Other Related Methods

Vector quantization methods discussed in the last section need not to be restricted to direct (spatial) image vector space. They are applicable to some transform image vector spaces as well. An example has been demonstrated in [28] in which a subimage is converted by a dual recursive *wavelet transform* to a set of wavelet coefficients and then this coefficient set, being regarded as a vector in the wavelet space, is quantized by the self-organizing feature map. Experiments show that the feature map yields comparable results to the LBG algorithm, and more importantly, the former is much faster than the latter.

Another neural-network-related image compression method operating in transform image vector space is recently proposed by Kong and Kosko (chap. 10 of [8]). Their network deals with vectors in the domain of *discrete cosine transform* (DCT). An image of size 256×256 is divided to, say, 256 subimages of size 16×16. Then the 16×16 two-dimensional DCT coefficients can be calculated for each subimage.† These DCT coefficients form a vector. An inverse DCT is used to convert the 16×16 DCT coefficients back to the original subimage of size 16×16 in the image domain. Using these DCT coefficients a so-called AC energy can be obtained.‡ In a conventional method, subimages are sorted according to their AC energy levels so that the 64 subimages with highest AC energy are classified to class 1, and those 64 subimages with less highest AC energy is classified to class 2, and so on [8]. There are totally four classes each of which contains 64 transformed subimages (vectors). After classification, four bit-allocating maps, one for each class, are used to scalar-quantize (encode) transformed subimages. The bit-allocating map for encoding subimages with higher AC energy (containing more image detail) has more bits, and while the one for encoding subimages with lower AC energy (containing less detail) less bits.

In arguing that the sorting of AC energy levels is computation-intensive, Kong and Kosko proposed an adaptive fuzzy association system that once it is trained by unsupervised *differential competitive learning*, it can determine the classification of a transformed subimage very quickly

---

† Let the $x(m, n)$ be an $N \times N$ subimage for $0 \le m,n \le N\text{-}1$, then the discrete cosine transform (DCT) is to compute the coefficient $X(u, v)$, for $0 \le u,v \le N\text{-}1$, by

$$X(u, v) = \frac{4c(u)\,c(v)}{N^2} \sum_{m=0}^{N-1} \left(\sum_{n=0}^{N-1} x(m, n) \cos \frac{(2m+1)\,u\pi}{2N} \cos \frac{(2n+1)\,v\pi}{2N} \right),$$

where $c(0) = 1/\sqrt{2}$ and $c(i) = 1$, for $1 \le i \le N\text{-}1$.

‡ AC energy is defined to be

$$\sum_{u=0}^{N-1} \left(\sum_{v=0}^{N-1} X^2(u, v) \right) - X^2(0, 0) .$$

[10]. In their system, certain characteristics of transformed subimages are extracted, for example, AC energy and lower-frequency AC energy (computed by replacing N by $N/2$ in the formula for AC energy). Also, possible classes for the subimages are predetermined, and in this case there are four classes: A, B, C, and D. A set of fuzzy rules are then compiled by associating all possible subimage characteristics with all possible subimage classifications. For example, one fuzzy association rule reads like

IF AC energy belong to region [*a, b*] and lower-frequency AC energy to region [*c, d*],
Then classify the subimage to class A.

If the whole range of AC energy is divided into 4 regions and the range of lower-frequency AC energy to 2 regions, there would be totally 4×2×4 = 32 fuzzy rules, for the last 4 due to four classes. These fuzzy rules are then embedded in a neural network to be trained via an unsupervised differential competitive learning scheme. The selection of proper regions for (lower-frequency) AC energy is derived from training vectors. And initial classifications of training subimages are provided by the conventional method mentioned earlier. After training, only a subset of all fuzzy rules will be selected by the network. The selected fuzzy rules can now be used to classify both trained and untrained subimages. The classified subimage will be scalar-quantized by one of the four bit-allocating maps mentioned earlier, depending on its classification. The major distinctions between the fuzzy system and the conventional method are that they classify subimages differently and the former is not restricted to classify the same numbers of subimages into each class. Furthermore, once the system is trained, any transformed subimage can be quickly classified without any information from other subimages. This has a potential for real-time image coding, or compression. The experiment results show that the fuzzy system is slightly better than the conventional method on both the SNR and the compression ratio. For a trained image, SNR = 25.72 dB is obtained at a compression ratio of 15.9, and for an untrained image, SNR = 24.56 dB at a compression ratio of 18.4.

The VQ methods in the last section concentrate on using neural networks to generate suitable codebooks. Some researchers, however, address a different issue of vector quantization via neural networks, namely, the way to efficiently encode input vectors given that the codebook is fixed or already trained. In [29] a tree-structured network has been proposed to encode vectors. Each node in the tree has a binary decision-making network which bipartitions the vector space. Starting from the root node, an input vector tested by the binary network is forwarded to either the left child or the right child of the root, and the winning child in turn makes its decision on to which of its two children it should forward the input vector. This process continues till the input vector gets to the bottom of the tree. As each node makes a binary decision on the input vector, the collection of these decisions can be used to encode the input vector. The advantage of such a tree-structured network is that after proper training, it has a time complexity of $\log_2 k$ for encoding a vector as compared to a time complexity of k for a VQ method with linear search, where k is the size of the codebook. (Note that the VQ methods discussed in the last section all have a subnet computing $\|x(t) - m_i(t)\|^2$, for all i, in parallel. And thus once these networks are trained, a simple network such as unsupervised LVQ may be used to encode a vector and its time complexity will then depend upon the settling time of the variable y_i.) Minimal results were given in [29]. One drawback of this network is that due to the fact it supports only linearly partition of the vector space, the selected reference vector may not be the nearest reference vector for a given input vec-

tor. An example has been considered in [30], for which case reference vectors are not linearly separable.

Two networks, multilayer feed-forward network and the Hopfield network, were used in [30] for vector encoding, given that the codebook is trained. Both networks were found to be able to separate linearly non-separable reference vectors. For the multilayer feed-forward network, it has a drawback that the number of hidden neurons must be very large in order to achieve a reasonably good performance. For the Hopfield network with the number of neurons equal to the number of reference vectors, it behaves very much like an unsupervised LVQ due to the network construction. At a high neuron gain, the network will select the reference vector nearest to an input vector.

Another neural-network-related image compression method has been recently proposed by Stark [31]. In fact, this method is more related to image decompression than compression. In that paper, neural networks have been used to decode the fractal images which are encoded by iterated function systems (IFS). (See [32] for details on fractal images and IFS.) The IFS code for an image contains a comparatively small set of real numbers which can be used to iteratedly generate the (approximately) original image. Because the iterations involved for fractal image generation are time-consuming, neural networks with one neuron per image pixel are constructed as a highly parallel architecture for fractal image rendering. This is done by viewing the image screen, where the fractal image to be displayed, as a plane of pixels (neurons) and exploring that the IFS code can be embedded to the neuron plane with finite numbers of weighted connections between neurons. Once the network is set up according to the IFS code of an image, all neurons evolve in parallel to generate the final fractal image in just few steps. In IFS theory a fractal image is regarded as a collection of *fixed points* or *limit points* of the corresponding IFS code, so initial values of neurons can be made arbitrarily as they do not effect the final scene. The drawbacks of these IFS networks are that they are size-dependent and thus non-scalable, and their connection weights need to be reprogrammed for every image. The latter is not an easy task as there will be 2^{18} weights in a network that renders fractal images of 512×512 pixels. Stark also proposed an neural network to compute the Hutchinson metric, a natural measure of the discrepancy between two images [33]. Since speed is one of the main concerns for fractal compression due to the involvement of iterative discrepancy measuring of subimages, such a network when properly incorporates into the fractal compression process can yield a significantly speedup to compression time.

6. Conclusion

Direct methods though have the advantage of straightforward implementation, but there is no theoretical results to, even asymptotically, guarantee the bound of the difference (MSE) of the original image and its decompressed image. Also, there is no direct interpretation of the compressed data obtained by those direct methods. One plausible interpretation is that those networks through training process learn to extract a high level abstraction of the interrelationship between the (sub)image pixels. But it is unlikely that more training images or more training iterations can improve the SNR for both trained and untrained images at the same time, as evidenced by the

results shown in Table 1. One way of enhancing the SNR as well as the compression ratio is to use networks with more hidden layers, but this will result in a significant increase of the network training time. Choices to reduce the network training time include using a faster training algorithm, such as the nested backpropagation scheme suggested in [13], or setting a part of connection weights fixed during training and only allowing the rest weights to adapt, which is exactly the case in [14]. When compared to VQ-based methods, direct methods, even with multiple-hidden-layer networks, are still inferior in that their SNRs for untrained images are much lower than trained images. Thus, as far as the image fidelity is concerned, direct methods are restricted to trained images compression.

Compared to direct methods, neural network based vector quantization methods are not bounded to trained images compression. From the experimental results obtained thus far, these methods attain comparable SNRs for trained and untrained images. Among these methods, the LVQ network and the self-organizing feature map are the favorable choices because they are easy to implement and render reasonable good SNRs. A better alternative, as far as a higher SNR is concerned, is the feature map with stochastic relaxation, but it demands a longer training period and more computations [27]. Using a larger codebook will always increase the SNR but at the sacrifice of compression ratio. One major advantage of most VQ networks is that they are directly hardware implementable via parallel VLSI architecture.

The fuzzy system combined with neural network learning scheme though suggest an alternative for image coding, but its improvement over the conventional methods is quite limited. This is due to the fact that the fuzzy system seeks to reclassify subimages among few available classes and such a reclassification saves only a few bits for one subimage, once the bit-allocating maps used to scalar-quantize each class of subimages are fixed.

Overall, though the results reported to date are far from conclusive, neural network based methods do provide other means for image compression, and there are instances where some of them yield better SNRs than the LBG algorithm. It is this author's view that vector quantization methods are potentially more profitable. This is because once they are implemented in parallel hardware architecture they can offer a direct search of the best match reference vector from the codebook in a constant time, as compared to the linear time needed in a exhausted search of an array codebook or the logarithmic time in a tree search of a tree-structured codebook. If, however, the desired codebook size is too large to permit a fully parallel hardware implementation for a direct search of the best match vector, a cascaded stages of, say, LVQ networks may be used. For such a cascaded network, at each stage a winning y_i is produced, and its corresponding reference vector is forwarded to the succeeding stage. The succeeding stage then determines its winning y_i and forward the corresponding reference vector to the next stage, and so on, until the overall winning reference vector is selected. The time complexity of such a cascaded VQ network is equal to the multiplication of the single-stage time complexity, which is a constant, and the number of stages.

Though neural network based vector quantization methods have the advantage mentioned above, it is unlikely that applying them directly to spatial image vectors can achieve the best compression ratio together with a reasonable SNR. Image pre-processing or domain transformation, such as DCT or wavelet transform, may generate transform image data that are more compress-

able via vector quantization. Much more work is needed on combining pre-processing or transformation with neural network based vector quantization. If the parallel nature of the neural network based vector quantization can be fully exploited, a hybrid of pre-processing/transformation and neural network based vector quantization may result in a more efficient paradigm for image compression.

Reference

[1] G. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30-44, April 1991.

[2] A. Maren, C. Harston, and R. Pap, Handbook of Neural Computing Applications, Academic Press, San Diego, CA, 1990.

[3] R. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

[4] G. Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, pp. 185-234, 1989.

[5] K. Knight, "Connectionist: ideas and algorithms," *Communications of the ACM*, vol. 33, no. 11, pp. 59-74, Nov. 1990.

[6] R. Hecht-Nielsen, Neurocomputing, Addison-Wesley, Reading, MA, 1990.

[7] J. Hertz, R Palmer, and A. Krogh, Introduction to the Theory of Neural Computation, Addison-Wesley, Redwood City, CA, 1991.

[8] B. Kosko, Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[9] G. Cottrell, P. Munro, and D. Zipser, "Image compression by back propagation: an example of extensional programming," Models of Cognition: A Review of Cognitive Science, N. E. Sharkey (Ed.), Ablex, Norwood, NJ, pp. 208-240, 1989.

[10] N. Sonehara, M. Kawato, S. Miyake, and K. Nakane, "Image data compression using a neural network model," *International Joint Conference on Neural Networks*, Washington, D.C., pp. II35-II41, 1989.

[11] R. Kohno, M. Arai, and H. Imai, "Image compression using a neural network with learning capability of variable function of a neural unit," *SPIE Vol. 1360 Visual Communications and Image Processing*, pp. 69-75, 1990.

[12] D. Anthony, E. Hines, J. Barham, D. Taylor, "A comparison of image compression by neural networks and principal component analysis," *International Joint Conference on Neural Networks*, San Diego, CA, pp. I339-I344, 1990.

[13] A. Namphol, M. Arozullah, and S. Chin, "Higher order data compression with neural networks," *International Joint Conference on Neural Networks*, San Diego, CA, pp. I55-I59, 1991.

[14] M. Mougeot, R. Azencott, and B. Angeniol, "Image compression with back propagation: Improvement of the visual restoration using different cost functions," *Neural Networks*, vol. 4, pp. 467-476, 1991.

[15] R. DeVore, B. Jawerth, and B. Lucier, "Data compression using wavelets: error, smoothness, and quantization," *Data Compression Conference*, Snowbird, Utah, pp. 186-195, 1991.

[16] R. Gray, "Vector Quantization," *IEEE ASSP Magazine*, pp. 4-29, April 1984.

[17] N. Nasrabadi and R. King, "Image coding using vector quantization: a review," *IEEE Transactions on Communications*, vol. 36, no. 8, pp. 957-971, August 1988.

[18] T. Kohonen, Self-Organization and Associative Memory, 2nd ed., Springer-Verlag, Berlin, Heidelberg, Germany, 1988.

[19] N. Nasrabadi and Y. Feng, "Vector quantization of images based upon the Kohonen self-organizing feature maps," *IEEE International Conference on Neural Networks*, San Diego, CA, pp. I101-I108, 1988.

[20] M. Naillon and J. Theeten, "Neural approach for TV image compression using a Hopfield type network," *Advances in Neural Information Processing Systems 1*, D. Touretzky (Ed.), Morgan-Kafman, San Mateo, CA, pp. 264-271, 1989.

[21] S. Ahalt, P. Chen, and A. Krishnamurthy, "Performance analysis of two image vector quantization techniques," *International Joint Conference on Neural Networks*, Washington, D.C., pp. I169-I175, 1989.

[22] S. Ahalt, A. Krishnamurthy, P. Chen, and D. Melton, "Competitive Learning algorithms for vector quantization," *Neural Networks*, vol. 3, pp. 277-290, 1990.

[23] J. Baras and A. LaVigna, "Convergence of Kohonen's learning vector quantization," *International Joint Conference on Neural Networks*, San Diego, CA, pp. III17-III20, 1990.

[24] T. Kohonen, "Improved versions of learning vector quantization," *International Joint Conference on Neural Networks*, San Diego, CA, pp. I545-I550, 1990.

[25] W. Fang, B. Sheu, and O. Chen, "A neural network based VLSI vector quantizer for real-time image compression," *Data Compression Conference*, Snowbird, Utah, pp. 342-351, 1991.

[26] T. Lee and A. Peterson, "Adaptive vector quantization using a self-development neural network," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 8, pp. 1458-1471, Oct. 1990.

[27] E. Yair, K. Zeger, and A. Gersho, "Competitive learning and soft competition for vector quantizer design," *IEEE Transactions on Signal Processing*, vol. 40, no. 2, pp. 294-309, Feb. 1992.

[28] M. Antonini, M. Barland, P. Mathieu, and J.C. Feauveau, "Multiscale image coding using the Kohonen neural network," *SPIE Vol. 1360 Visual Communications and Image Processing*, pp. 14-26, 1990.

[29] E. Wan, P. Ning, and B. Widrow, "Neural tree structured vector quantization," *International Joint Conference on Neural Networks*, Washington, D.C., pp. II267-II270, 1990.

[30] M. Naraghi-Pour, M. Hegde, and F. Bourge, "A comparison of to neural network architectures for vector quantization," *International Joint Conference on Neural Networks*, San Diego, CA, pp. I391-I396, 1991.

[31] J. Stark, "Iterated function systems as neural networks," *Neural Networks*, vol. 4, pp. 679-690, 1991.

[32] M. Barnsley, Fractals Everywhere, Academic Press, New York, 1988.

[33] J. Stark, "A neural network to compute the Hutchinson metric in fractal image processing," *IEEE Transactions on Neural Networks*, vol. 2, no. 1, pp. 156-158, January 1991.

Neural Networks for Image Compression: A Survey

Research Memo # DRD-92110

Chia-Yiu (Matthew) Maa

Electronic Data Systems
Detroit Research and Development
3551 Hamlin Rd.
Auburn Hills, Michigan 48326

April 9, 1992

ABSTRACT

This report surveys the neural network related methods being used for image compression. These methods can be categorized into two major groups, direct methods and vector quantization methods. Direct methods mostly adopt three-layered, feed-forward networks in which input and output layers have the same number of neurons and hidden layers have fewer neurons. These networks are trained via certain pre-selected learning schemes during the training stage so as to generate at output layers (approximately) the same images that were given to input layers. The first halves of these networks, from input layers to hidden layers, are encoders (compressors), while the second halves, from hidden layers to output layers, are decoders (decompressors). The outputs of neurons of hidden layers are considered compressed images since hidden layers have fewer neurons. For vector quantization methods, networks are used to generate a proper codebook of reference vectors, where a vector is a subimage. A subimage can then be encoded by using the index of its best-match reference vector in the codebook. Compression is achieved as a vector index having a shorter binary representation than the vector itself. Other related image compression methods, including fuzzy systems and iterated function systems as implemented by neural networks, will also be addressed. All methods surveyed in this report are lossy, that is, they only render from compressed files an approximate image to the original.