Image Compression Using a Neural Network

S P Luttrell
Royal Signals and Radar Establishment, St Andrews Rd, Malvern, WR14 3PS, UK

Data compression of speckled images poses a non-trivial model identification problem. We train an unsupervised neural network on a set of archetype images in order to form an internal representation (or model) of the image features. We find that a multi-layer topographic mapping network has the necessary properties successfully to compress and reconstruct imagery. We show how to extend and improve upon existing learning algorithms for this type of network, and we express the network learning dynamics as a diffusion equation. We then present some examples of the application of this technique to synthetic aperture radar images.

Keywords: neural network, topographic mapping, synthetic aperture radar, speckle.

I. INTRODUCTION

Successful data compression requires knowledge of the source statistics [1]. We shall be concerned with the compression of coherent images (specifically synthetic aperture radar (SAR) images) for which there do not exist any complete a priori statistical models. In a recent study [2] a conventional predictive compression system was used with some success on SAR imagery. In this paper we shall present a neural network solution to the same problem.

The main difficulty which is encountered when constructing a data compression system is identifying the correct data model. There has been much interest recently in neural networks [3] because they can be trained to form an internal representation (model) of structure in the training set. Such a model is non-trivial in the sense that the neural network forms a non-linear mapping from data space to its internal space, so it may extract complex features from the data. A neural network approach to model identification is ideal in situations where we have little or no a priori knowledge, but we have available a large amount of training data.

Model identification is a particular form of inverse problem in which we must recover the underlying rules and regulations (e.g. physical laws) which generate data sets. This inverse problem must be solved as a prelude to the conventional inverse problem of recovering the cause (e.g. placement of scatterers) of a particular data set. Various approaches are used to solve this problem in SAR image analysis, ranging from the use of generalised noise processes to derive clutter models [4], to a statistical analysis of a number of images to extract a heuristic image model [5]. In each case the large number of image pixels is "explained" in terms of a much smaller number of causal factors.

Conventionally in image processing [6] a feature space of plausible image measurements (or statistics) is defined, and their means and covariance matrices for different image classes are used to extract a feature subspace suitable for solving whatever problem one has in mind. In particular, the prediction of the next pixel value in a row of a SAR image based on knowledge of features which depend only on previous pixel values would provide a solution to the SAR image compression problem. For this purpose we seek that set of features which is maximally correlated with the next pixel value, but which at the same time has minimal internal redundancy. This interpretation emerges naturally in the maximum entropy approach of [5].

On the other hand, neural network processing of images allows feature spaces to be crafted ab initio based on a set of representative images. The advantage of such networks is their adaptability when faced with the task of constructing feature spaces with subtle properties. Neural networks may be trained both in a supervised and an unsupervised fashion. For instance, the multilayer perceptron (supervised) may be used to distinguish between sonar returns from a rock and a metallic cylinder on the seabed [7]. For texture modelling, the maximum entropy approach in [3] leads to a rudimentary neural network model expressed as a Markov random field.

It is sometimes argued that a neural network model is not based on physical deductive reasoning, and is therefore a mere collection of ad hoc observations with limited predictive power. However, neural networks (like us!) turn out to be remarkably devious in their ability to reduce complicated structure to a few simple rules. This is very much in the spirit of physical model construction, so we find their use entirely palatable.

We shall concentrate on unsupervised neural networks because we wish to compress images for later analysis (and/or reconstruction) without prejudicing the information content of the feature space by supervising its formation. Typically supervised training leads to feature spaces which preserve only part of the information which is required to reconstruct an image. On the other hand unsupervised training has to be non-committal about what information it preserves, so it tends to preserve some information about everything.

In Section III we introduce a neural network structure which is suitable for compressing images, and in Section...
III. we present a theoretical analysis of the properties of such networks. In Section IV we present the results of some experiments on SAR images.

II. A NEURAL NETWORK

The data compression problem is one of discovering a mapping \( T \) which takes an input \( x_{\text{in}} \) into an output \( x_{\text{out}} \) (both usually vector-valued) in such a way that a pseudo-inverse \( T \) exists which performs the inverse mapping with at most too much distortion. Thus

\[
\begin{align*}
T x_{\text{in}} &= x_{\text{out}} \\
T x_{\text{out}} &= x_{\text{in}}
\end{align*}
\]  

(2.1)

where \( D(x_{\text{in}}, x_{\text{out}}) \) is a distortion measure, angle brackets denote an average over \( x_{\text{in}} \), and \( D_0 \) is some maximum acceptable average distortion. Furthermore data compression is achieved only if \( x_{\text{out}} \) occupies fewer bits of memory than \( x_{\text{in}} \), where we ignore the fixed overhead used to store \( T \) and \( T^{-1} \). For image compression \( x_{\text{in}} \) is a \( d_{\text{in}} \)-dimensional vector of pixel values each component of which is represented by a small number of bits (usually 8), and \( x_{\text{out}} \) is a \( d_{\text{out}} \)-dimensional coded image. However \( d_{\text{in}} \) is usually enormous (\( 10^6 \) is not uncommon), so the compression mapping \( T \) can operate on at most a very small proportion of the image pixels at a time: the mapping must be factorised.

A good review of vector quantisation (applied to the compression of speech) may be found in [8], and the network structure presented in [9] provides a means of adaptively designing vector quantisers. We have adopted an analogous approach in order to compress images, but we find that several extensions of the basic method of [9] are necessary in order to obtain a satisfactory real-time compression/reconstruction performance when \( d_{\text{in}} \) is large.

A vector quantiser mapping \( T \) is specified as follows

\[
T x_{\text{in}} = x_{\text{out}} \equiv k_0
\]  

(2.2)

where \( d(x_{\text{in}}, x_{\text{in}}^{k_0}) \) is a suitable metric, \( k_0 \) minimises \( d(x_{\text{in}}, x_{\text{in}}^{k}) \) w.r.t. \( k \), and the \( x_{\text{in}}^{k} \) form a set of \( n \) input code vectors against which \( x_{\text{in}} \) is compared. When \( d(x_{\text{in}}, x_{\text{in}}^{k}) \) is Euclidean the \( x_{\text{in}}^{k} \) partition the input space into \( n \) convex polyhedra whose surfaces lie on the hyperplanes \( d(x_{\text{in}}, x_{\text{in}}^{k}) = d(x_{\text{in}}, x_{\text{in}}^{j}) \) \( j \neq k \). The pseudo-inverse mapping \( T \) is then given by

\[
T x_{\text{out}} = x_{\text{in}} \equiv x_{\text{in}}^{k}
\]  

(2.3)

The task facing the neural network is to learn an optimal set of \( x_{\text{in}}^{k} \) which minimises some average over \( x_{\text{in}} \) of the reconstruction distortion \( D(x_{\text{in}}, x_{\text{in}}) \). It turns out that a further desirable property may easily be obtained from a neural network, namely that \( x_{\text{in}} = T x_{\text{out}} \) is a continuous function of \( x_{\text{out}} \). This type of mapping is called topographic because it attempts to preserve the topological structure of the input space in its coded output space representation.

A. The network learning algorithm

[7] specifies the following type of algorithm for adaptively altering the \( x_{\text{in}}^{k} \)

\[
x_{\text{in}}^{k}(t+1) = x_{\text{in}}^{k}(t) + \epsilon(|k - k_0|, t) (x_{\text{in}} - x_{\text{in}}^{k}(t)) \\
|k - k_0| \leq r_{\text{max}}(t)
\]

(2.4)

where \( k_0 \) minimises \( d(x_{\text{in}}, x_{\text{out}}^{k}) \) with respect to \( k \), \( \epsilon(|k - k_0|, t) \) is an update scale factor which depends both on the number of prior adaptation cycles \( t \) and the output space separation \(|k - k_0|\), and \( r_{\text{max}}(t) \) specifies the maximum range (in output space) over which a particular \( k_0 \) may exert its updating influence. Both the update scale factor and the maximum range are monotonically decreasing functions of \( t \) [7].

Physically the update equations may be regarded as modelling the motion of a damped stiff hypersheet (specified as a mesh of code vectors \( x_{\text{in}}^{k} \)) in the \( d_{\text{in}} \) dimensional input space. The external force term generates the update step size, and the stiffness derives from the finite range \( r_{\text{max}}(t) \), which also leads to the topographic property.

B. Factorised mappings and lookup tables

For image processing we must factorise \( T \) in order to break up \( x_{\text{in}} \) into low dimensional pieces. We choose to break \( x_{\text{in}} \) into non-overlapping contiguous rectangular pieces of image, and factorise \( T \) into a corresponding product of identical mappings. For simplicity we shall henceforth use the full image notation \( x_{\text{in}} \) to describe a single rectangular region, unless otherwise stated. The heavy computational demands of searching for the vector \( x_{\text{in}}^{k} \) which minimises \( d(x_{\text{in}}, x_{\text{in}}^{k}) \) for each image rectangle are removed if we implement the mapping \( T x_{\text{in}} = x_{\text{out}} \) in lookup form. However this, in turn, places a tight upper bound on the number of bits \( A \) which we are permitted to use to represent \( x_{\text{in}} \), because \( 2^A \) is the size of the address space of the lookup table. We shall assume that both \( x_{\text{in}} \) and \( k \) are represented using 8 bits, so \( d_{\text{in}} = 2 \) is the only feasible choice given current memory technology. Whilst a worthwhile image compression can be obtained with such a small value of \( d_{\text{in}} \), we would prefer to use a larger value in order to make use of any image redundancy which exists within multiplets of three or more pixels.

Further compression is made possible by iterating the coding procedure to produce an encoded version of the previous output, and so on. In general such an iteration is not permitted unless the vector quantisers are topographic (as they are here). Non-topographic mappings lead to intermediate encodings of the image which do not have a simple metric structure, so they cannot be further encoded by a simple vector quantiser.
C. Improved learning algorithm

We shall now present an improvement in the learning algorithm. The original algorithm [9] keeps the number n of code vectors \(x_{in}^k\) fixed throughout the training process. This leads to long training times (tens of thousands of code vectors) and moderately large number of code vectors. The original algorithm [9] keeps the number n of code vectors. This causes long run times in the initial stage of the simulation. We choose to replace the original learning algorithm by one in which a small value of n is used initially to simulate only the long range structure of the sheet, and then n is progressively increased as t increases.

We find that the following changes to Equation 2.4 (here expressed only for \(d_{int} = 1\)) yield a highly effective scheme for image compression (and many other purposes).

\[
\begin{align*}
    r_{\text{max}}(t) &= 1 \\
    \epsilon(0, t) &= \epsilon_0 \\
    \epsilon(1, t) &= \epsilon_1 \\
    n &\rightarrow n(t) \in \{2, 3, 5, \ldots, 2^{i+1}, \ldots\} \\
    n(0) &= 2
\end{align*}
\]  \tag{2.5}

We step through the set of n(t) values by inserting a code vector \(x_{in}^k\) in half-way between each pair of adjacent existing code vectors. Thus

\[
x_{in}^k = (x_{in}^k + x_{in}^{k+1})/2 \quad k = 1, 2, \ldots, n - 1
\]  \tag{2.6}

and then relabel all the code vectors into the range \(1, 2, \ldots, 2^n - 1\). In order to minimise the amount of redundant computation we insert code vectors only when the n old code vectors have converged to a random motion commensurate with the update noise level due to the non-zero values of \(\epsilon_0\) and \(\epsilon_1\).

The monotonically decreasing values of \(\epsilon([k - k_0], t)\) and \(r_{\text{max}}(t)\) together with a fixed value of n which were used in Equation 2.4 correspond to the fixed values of \(\epsilon\) and \(r_{\text{max}}\) together with a monotonically increasing value of n used in Equation 2.5. Equation 2.5 merely block renormalises the degrees of freedom of the stiff sheet to concentrate on the large scale structure (in \(x_{in}\)-space) of the training set before proceeding to ever finer structure. For the image compression problem convergence to \(n = 257\) code vectors occurs in a few seconds (on a 5 MIPS MV20000).

III. AN ANALYTIC MODEL

It is instructive to derive a simple analytic model which allows us to explore the properties of a vector quantiser without actually running simulations. To this end we shall introduce two new quantities \(P(x_{in})\) and \(\rho(x_{in})\) which are, respectively, the probability density function (PDF) of the training images, and the density in \(x_{in}\)-space of the code vectors. \(\rho(x_{in})\) is necessarily an approximate quantity because there is only a finite number n of code vectors. These quantities are normalised as

\[
\begin{align*}
    \int dx_{in} P(x_{in}) &= 1 \\
    \int dx_{in} \rho(x_{in}) &= n
\end{align*}
\]  \tag{3.1}

Denote as \(S_k\) the set of \(x_{in}\) which map to \(k\)

\[
S_k = \{x_{in} : T x_{in} = k\}
\]  \tag{3.2}

By definition we recover \(x_{in}^k\) from \(\rho(x_{in})\) by integration

\[
x_{in}^k = \frac{\int dx_{in} \rho(x_{in}) x_{in}}{\int dx_{in} \rho(x_{in})}
\]  \tag{3.3}

The updates used in both Equation 2.4 and Equation 2.5 cause \(x_{in}^k\) to be moved by an amount proportional to \(x_{in} - x_{in}^k\) for each \(x_{in}\) in \(S_k\). From Equation 2.4 (or Equation 2.5) for fixed \(t\) and \(k = k_0\) the average update step induced on \(x_{in}^k\) is given by (ignoring the constant of proportionality)

\[
\delta x_{in}^k = \int dx_{in} P(x_{in}) (x_{in} - x_{in}^k)
\]  \tag{3.4}

The restriction to \(k = k_0\) destroys the topographic property but it leads to essentially identical results for \(\rho(x_{in})\). Using the Equation 3.3 we may rewrite Equation 3.4 as

\[
\delta x_{in}^k = \frac{\int dx_{in} dy_{in} P(x_{in}) \rho(y_{in}) (x_{in} - y_{in})}{\int dx_{in} \rho(x_{in})}
\]  \tag{3.5}

Now assuming that \(P(x_{in})\) and \(\rho(x_{in})\) are slowly varying functions of \(x_{in}\) in comparison with the size of \(S_k\), we may approximate them using the first two terms of a Taylor expansion about any suitable point in \(S_k\) (\(x_{in} = x_{in}^k\), say). Thus

\[
P(x_{in}) = P_0 + \nabla P_0 \cdot (x_{in} - x_{in}^k) + \cdots
\]

\[
\rho(x_{in}) = \rho_0 + \nabla \rho_0 \cdot (x_{in} - x_{in}^k) + \cdots
\]  \tag{3.6}

where the subscript 0 is used to denote evaluation at \(x_{in} = x_{in}^k\). For clarity let us define

\[
x = x_{in} - x_{in}^k
\]

\[
y = y_{in} - y_{in}^k
\]

\[
S_{k0} = \{x : x + x_{in}^k \in S_k\}
\]  \tag{3.7}

whence from Equation 3.5, Equation 3.6 and Equation 3.7 we obtain an expression for the \(i\)th component \(\delta x_{in,i}^k\) of \(\delta x_{in}^k\)

\[
\delta x_{in,i}^k \equiv \frac{\delta_i + \delta_i^{(2)} + \delta_i^{(3)}}{\nu}
\]  \tag{3.8}
where
\[ \delta_1^1 = P_0 \rho_0 \int \frac{dx \, dy}{S_k} (x_i - y_i) \]
\[ \delta_1^2 = P_0 \int \frac{\partial \rho_0}{\partial x_j} \int \frac{dx \, dy}{S_k} (x_i - y_i) y_j \]
\[ \delta_1^3 = \rho_0 \sum_{j=1}^{dk} \frac{\partial \rho_0}{\partial x_j} \int \frac{dx \, dy}{S_k} (x_i - y_i) x_j \]
\[ \nu \equiv \rho_0 \int \frac{dx}{S_k} + \sum_{j=1}^{dn} \frac{\partial \rho_0}{\partial x_j} \int \frac{dx \, dy}{S_k} \]

(3.9)

The \( \delta_1 \) term vanishes by symmetry which removes all leading order terms from the numerator, so the derivative term in \( \nu \) may now be discarded. The \( x, y \) parts of \( \delta_1^2 \) and \( \delta_1^3 \) are small compared with the \( x, y \) parts, and so may be discarded. Equation 3.8 and Equation 3.9 thus simplify to

\[ \delta x_{in,i}^k \simeq \sum_{j=1}^{dn} \left( \int dx x_i x_j \right) \left( \frac{\partial P_0}{\partial x_j} - P_0 \frac{\partial \log(\rho_0)}{\partial x_j} \right) \]

(3.10)

The integral in Equation 3.10 yields the inertia tensor for the volume \( S_k \) with \( x_{in}^k \) used as origin. If we assume that the isotropic component is dominant (i.e. approximately proportional to \( \delta_{1,j} \)) then

\[ \delta x_{in,i}^k \propto \nabla P_0 - P_0 \nabla \log(\rho_0) \]

(3.11)

\[ = P_0 \nabla \log \left( \frac{P_0}{\rho_0} \right) \]

We may define a current \( J \) as

\[ J \equiv \rho \nu \]

(3.12)

\[ \propto \rho P \nabla \log \left( \frac{P_0}{\rho_0} \right) \]

where \( \nu \) is a velocity. This leads to the diffusion equation

\[ \frac{\partial \rho}{\partial t} = -\nabla \cdot J \]

(3.13)

for the dynamics of the code vector density under the influence of updates of the type given in Equation 2.4 In equilibrium \( J = 0 \) whence \( P(x_{in}) \propto \rho(x_{in}) \). An argument which leads to this result for \( d_{in} = 1 \) has been presented in [9]. The \( x_{in}^k \) thus concentrate in regions where \( P(x_{in}) \) is largest. Now define the output probabilities as \( p_k \), then in equilibrium

\[ p_k \simeq \frac{P(x_{in}^k)}{\rho(x_{in})} \]

(3.14)

\[ \simeq \text{constant} \]

associated with reconstructions \( x_{in} \) from an original \( x_{in} \) is \( \frac{1}{\rho(x_{in})} \). Define an average logarithmic distortion as

\[ D \equiv \int dx \rho(x_{in}) \log \left( \frac{1}{\rho(x_{in})} \right) \]

(3.15)

Introducing a Lagrange multiplier to hold \( \int dx \rho(x_{in}) \) constant, and then functionally differentiating yields

\[ \frac{\delta D}{\delta \rho(x_{in})} = \int dx \left( \frac{P(x_{in})}{\rho(x_{in})} - \lambda \right) \delta(x-x_{in}) \]

(3.16)

which leads to a stationary solution \( P(x_{in}) \propto \rho(x_{in}) \) as required. Just as we have related the update prescription in Equation 2.4 to the distortion optimisation in Equation 3.16, we may relate other update prescriptions to other optimisations, and vice versa.

IV. APPLICATION TO SAR IMAGES

We have implemented a version of the multi-layer topographic mapping network which uses \( d_{in} = 2 \), \( d_{out} = 1 \) at each stage. For training we use \( c_1 = 0.1 \) and \( c_2 = 0.02 \) with 32 updates between each episode of code vector insertion. To assist the formulation as a table look-up scheme we iterate until \( n = 257 \) and then discard 1 vector (followed by re-equilibration) in order to obtain 256 code vectors which may be indexed using 8 bits. We also represent the input image using 8 bit pixels. Thus each stage of the hierarchical mapping is represented in a table with \( 2^{16} \) byte-sized entries.

A suitable choice of which pairs of pixels to use as \( x_{in} \), may be arrived at by physical argument. SAR images have a lot of short length scale correlations which can be used to immediate advantage in image compression. Thus the first stage of the mapping should code adjacent pairs of pixels. Furthermore, the SAR imagery which we have used has a poorer range than azimuth resolution, so we choose the pairs of pixels to lie adjacent in the range direction to take advantage of the extra range correlations. The choice of pixel pairs at later stages of compression is less clear cut, so we have chosen to alternate the initial prescription between the range and azimuth directions - flipping once per stage of the hierarchical mapping. Thus a 2 stage mapping will compress by a factor of 2 in each of the range and azimuth directions, and a 3 stage mapping will compress by a further factor of 2 in the range direction.

We start with autofocus SAR images with a resolution of 1.5m. We then average the moduli of all contiguous non-overlapping \( 2 \times 2 \) pixel blocks to produce a smoother image (with \( \frac{1}{4} \) of the number pixels of the original image). This is similar to multilook imaging, and it reduces the effect of speckle without losing much significant image detail.

Figure 1a a 256 \times 256 SAR image of a urban region produced in this way. Figure 2, Figure 3, and Figure 4 show
the reconstructions of Figure 1 which were obtained after compression by factors of 2, 4 and 8 respectively. Visually Figure 2 and Figure 3 are very accurate renditions of the original, whilst Figure 4 is found somewhat wanting. A quantitative comparison of the reconstructions with the original can be found in [10].
We also processed a SAR image of a rural region. Figure 5, Figure 6, Figure 7, and Figure 8 are analogous to Figure 1, Figure 2, Figure 3, and Figure 4 except that the code vectors used were those derived from Figure 1 (not Figure 5). The quality of the reconstructions demonstrates that the data compression is not sensitive to the precise details of the image statistics. This allows us to use a single set of look-up tables to compress a wide variety of SAR images, thus avoiding the associated storage overhead.
V. CONCLUSIONS

We have shown that a multi-layer topographic neural network may successfully be used to compress SAR images by factors of 2, 4 and 8. The formulation of the multi-layer mapping as a set of table look-up operations leads to extremely fast compression and reconstruction of SAR images, so the technique could be used in real-time applications.

Once the basic network software has been written no further significant effort is required because model identification occurs automatically as the neural network is trained. The same software could, in principle, be used to learn suitable codings for other types of imagery. The most pleasing result is the fact that the neural network has proved itself to be capable of unsupervised learning of a non-trivial mapping from data space into an internal representation space.