Colour and Texture in Cloud Identification: An Experimental Comparison of Neural Network and Bayesian Methods.

K. Richards and G.D. Sullivan

Intelligent Systems Group,
Department of Computer Science
University of Reading, RG6 2AY, U.K.
K.Richards@reading.ac.uk.

Abstract

The employment of Artificial Neural Networks to the classification of meteorological data has been considered in previous papers and found to offer promising results. We compare the performance of the Bayesian Classifier with two different Neural Network architectures. The classifiers were used to segment images of cloud into four different meteorological classes on the basis of spectral-textural measurements.

The experimental design is based on a set of 60 hand-classified images, random sampling of which enables us to generate training and test sets. This design allows us to carry out a repeatable comparison of the classifiers with different training and test data.

1 Introduction.

In the course of developing an automatic ground-based cloud observation system, we have studied a number of possible methods to extract useful spectral (colour), and textural (local spatial) features. The spectral-texture features used are an extension of those used by Laws, from monochrome to colour images. A set of local linear transforms is applied to each spectral channel (RGB), and a macro-statistic taken over a 32x32 window [4,6]. Larger neighbourhood windows have been found to offer no improvement in classification, but introduce greater errors at boundaries between classes.

The image data consists of a set of ground-based images of the sky captured over a period of two weeks using a colour CCD camera and digitiser. As part of a collaborative study, the Meteorological Office provided a trained observer to classify the images into regions of Cumulus, Cirrus, Sky and Stratus. Of approximately 130 classified images, 60 of the most representative were selected for this study. These images include examples of the four classes taken at various times of day, and at various angles with respect to the sun.

This paper reports an experiment to compare two Neural Network classifiers, and a commonly used statistical approach, under as near operational conditions as we can obtain. Random samples of the image set were used as training data with the remaining images used for testing. This allowed a repeatable trial of the classifiers with different image sets.

2 Neural Networks.

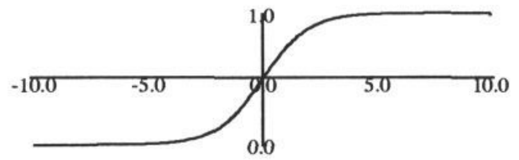
Neural Networks have already been employed in the automatic classification of meteorological cloud observation data, although to date this has invariably taken the form of satellite data. Some of these experiments have compared Neural Networks against statistical methods, such as KNN, Euclidean distance classifiers, Maximum Likelihood classifiers, and have reported comparable if not superior performance [1,5,8].

Previous studies have usually concentrated on a limited set of samples taken from a single set of satellite images. This strategy is adequate for satellite images, which contain relatively little inter-image variation, but ground-based data such as ours, is far more variable. The position of the sun, the time of day, and the angle with which the image was taken, all result in a very wide range of lighting conditions. The experimental design uses test and training data randomly taken from within a common set of images. Here we preselected independent sets of test and training images to study the abilities of the classifier in the face of both inter- and intra-image differences.

The Neural Networks we have used in this experiment are the standard Back-Propagation Neural Network (BPNN), and a Counter Propagative Network (CPN), both of which are well studied Neural Network architectures.

2.1 BPNN Architecture.

Neural Networks consist of a network of interconnected processing elements, with each element having its own set of inputs and associated weights. When in operation a cell calculates its response by summing the product of all inputs with their weights. A non-linear transfer function is applied to this sum, giving the actual output from the cell. The transfer function may be a simple threshold, or a function which simulates the non-linear characteristic of biological cells. Functions such as the Sigmoidal Logistic function (Figure 1), and the Hyperbolic Tangent function (tanh), are frequently used. The necessity for the non-linearity, and the potential for representing arbitrary functions with multiple layer networks is described in many introductory Neural Network text books such as Hecht-Nielsens [2] or Wassermans [9].



$$SIG = \frac{1}{(1 + exp(-SUM))} \qquad (1)$$

Figure 1 : The sigmoid response function.

A standard artificial Neural Network configuration consists of three types of layer: an input layer, an output layer, and one or more intermediate hidden layers (see Figure 2). In operation, a set of inputs is presented at the input layer, and consecutive layers of the network are processed, with each layer's output forming the input to the following layer. The final layer provides the network's output. Alternative architectures such as recurrent networks, or networks with connections which skip layers have been suggested but are outside the scope of this paper.

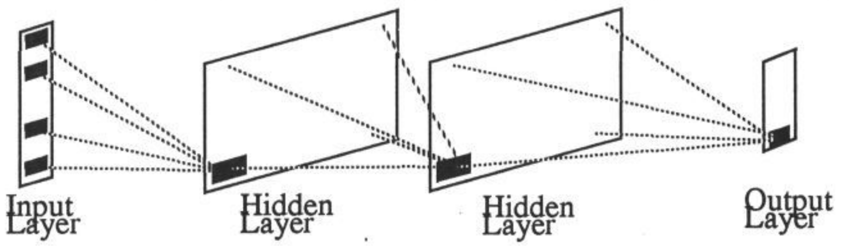


Figure 2 : Neural Network consisting of four layers.

The principal attraction of NNs is the ability to train the behaviour of a system in response to data of known origin. A number of Neural Network training schemes, both supervised and unsupervised, have been proposed.

2.2 Backpropagation Training.

Using backpropagation the error of the entire network to a given input is calculated at the output layer, and each cell is given its own error term δ_{output}. Usually for binary outputs this is just the difference between the target and the output (see Equation 2).

$$\delta_{output} = Target - Output \qquad (2)$$

This error is passed back through the input weights to the previous layer. Each cell in this layer sums all of the errors from the cells in the output layer. This sum, multiplied by the derivative of the transfer function, represents the error of a hidden cell δ_{hidden} (Equation 3). This value will be subsequently passed *back* to other hidden layers, and so the error is *propagated.*

$$\delta_{hidden} = Out_{hidden}(1 - Out_{hidden}) \sum_{i=1}^{N} \delta_i \cdot W_{hidden,\,i} \qquad (3)$$

An identical weight updating process is used in both the hidden and output layers. The *generalised delta rule* learning law (see Equation 4), is guaranteed to converge to a minimum error state (where α, *is the learning coefficient, l* refers to the current layer, *i* refers to the current cell, *j* refers to the current input cell, δ_{li} refers to the error of cell i in layer 1, and O_{li} is the output of cell *i* in layer *l*). Although this learning law always minimises the error, it takes an indefinite length of time to converge. For practical purposes a finite version of the rule, with a fixed batch size N, is often used.

$$W_{lij}^{new} = W_{lij}^{old} - \alpha \lim_{N \to \infty} \sum_{k=1}^{N} \delta_{li}^{k} O_{(l-1)j}^{k} \qquad (4)$$

There are many possible variants on this basic architecture and learning law with various advantages and pitfalls. Unfortunately there is no clear mechanism for choosing between them, and the choice of learning law, parameters, and architecture must be selected to suite each particular problem.

2.3 CPN Architecture.

The second NN architecture we have looked at is the Counter-propagative neural network. It is a combination of two architectures, Kohönen's self-organising topographic map, and Grossberg's "*Instar/Outstar*" structure [9]. Hecht-Nielsen developed the CPN in order to use Kohönen's self organising map to learn explicit functions [2].

We have used the CPN in its *forward-only* variant. The learning phase of the Kohönen layer partitions the input space into clusters of similar input vectors. The Grossberg layer is then trained to map the Kohönen Layer onto the desired classifications (see Figure 3). When fully trained it functions as a near-optimal lookup table. During training the input vector consists of both the input data X, and the target output Y. The network is trained to produce the same output as

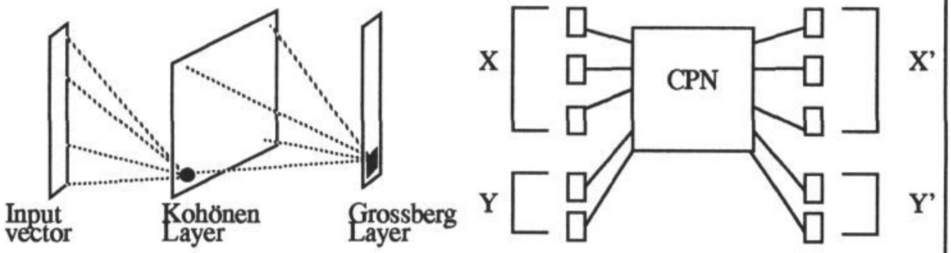
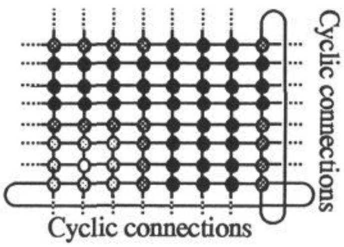


Figure 3 : The CPN: During training both the class and input data are presented to the network. The Network is trained to produce the same output as input. After training only the input data needed to be supplied to obtain it's associated class.

input, building an association between the data and its class. When fully trained, the input data alone is sufficient to identify its class. The simplicity of this training mechanism has made the CPN very attractive for problems such as image compression.

2.4 Training the Kohönen Layer.

A Kohönen Layer consists of an array of cells organised in a 2-D spatial array (although higher dimensional structures are also possible). The array is cyclic, so its local neighbourhood extends across borders, as shown Figure 4, where each shade of grey represents a constant *cyclic* distance from the white cell.



$$CyclicDistance = min(dx, Width - dx) + min(dy, Height - dy) \quad (5)$$

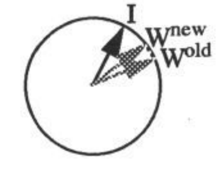
where

$$dx = abs(X^1 - X^2) \quad (6)$$
$$dy = abs(Y^1 - Y^2) \quad (7)$$

Figure 4 : The Kohönen layer. The local neighbourhood connections are cyclic.

Each cell in a Kohönen layer has its own normalised weight vector, which is initialised to a set of random values at the start of training. The response of a cell is simply the dot product of this weight vector with the input vector (see Figure 5). The Kohönen layer works on a "*winner takes all*" process. After all the cell's responses are calculated, the best responding cell has its output set to 1, and all the remaining cells are set to 0.

Kohönen's learning law considers the input vector and the weights vector to represent points on a hyper-sphere (see Figure 5). Adjustment of the weights to improve the match with the input vector simply involves the addition of some fraction of the difference between the input vector to the original weight vector. This moves the weights closer to the input vector. The fractional adjustment of the weights is controlled by a learning parameter α.



Response $$Z_i = W_i \bullet I \qquad (8)$$

Kohönen's Learning Law

$$W_i^{new} = W_i^{old} + \alpha \times (I - W_i^{old}) \times Z_i \qquad (9)$$

Figure 5 : Training the Kohönen layer. The winning cell and its neighbours all have their weights adjusted to look more like the input vector.

During training, weights are modified in a local neighbourhood around the winning cells. Initially this neighbourhood covers virtually the entire network, but as training proceeds the size of this neighbourhood is reduced until only the single winning cell's weights are changed. By this use of *lateral excitation* a smooth topological map of the input space is formed, and the feature space is distributed across the entire network. Variants on the Kohönen learning mechanism, such as cell conscience or a local neighbour bias, can ensure a homogenous network [2].

2.5 Training the Grossberg Layer.

As described earlier the Kohönen layer will have one cell responding. The Grossberg layer contains a set of weights connecting its cells, to the outputs of the Kohönen layer. If the output is non-zero then the Grossbergs weights are changed. The amount of adjustment is proportional to the difference between the weight vector and the desired output. In the case of the CPN this is the original input vector.

$$Output_{ij} = Output_{ij} + \alpha\,(Input_j - Output_{ij}) \qquad (10)$$

The weight connecting Grossberg cell j to Kohönen cell *i* is adjusted to look more like the input (as with the previous training mechanisms, a learning parameter α is used). The *weights* should converge to the average values of the desired outputs, for all the input vectors which cause the Kohönen cell to fire, thus the Kohönen layer generates clusters in the feature space, the purpose of the Grossberg layer is to map these clusters into the desired outputs.

3 Bayesian Classification.

The performance of the two Neural Networks were compared to that of a Bayesian classifier, which is widely used to classify multi-source data. Although it makes the assumption that the class distributions are multi-normal, it can be used successfully without any guarantee of the normality of the feature distributions [7,10]. The Bayesian classifier uses a class distance measure based upon the class mean and the class feature covariance

$$Bdist_i = ((f - v_i) \bullet C_i^{-1} \bullet (f - v_i)) + log(|C_i|) \quad [i=1....N] \qquad (11)$$

where f is the vector to be classified and v_i and C_i are the mean feature vector and covariance matrix for class i. When a distribution is multi-modal, or is heavily skewed, the Bayesian Classifier is expected to perform poorly, and other *non-linear* approaches such as Knn classifiers and Neural Networks may perform better [1,5].

4 Random Sample Trial.

Ten random sample sets were created from the original set of 60 images, with each sample consisting of 30 images. Each set provided training data for the classifiers, with the remaining images used for testing the classifier's performance.

The images were processed to extract spectral-texture features similar to Laws Texture measures [4]. Each feature is the local energy of one of nine 3x3 linear transforms, applied to each colour channel separately. This simple colour extension to the monochromatic texture analysis used by Laws [4,6], provides a total of 27 features. Other researchers such as Unser [7], have used Principal Component Analysis to identify sets of local linear transforms with better discriminating power, however similar experiments with our data showed no apparent improvement in classification.

Since there were limited examples of cirrus available in the data set we biased the random samples to ensure that each training set contained some examples of cirrus. The number of samples for each image was also biased so that the relative number of examples for each class would be the same. This limited selection of cirrus images accounts for much of the poor classifier performance. Future data collection exercises will concentrate largely upon increasing the number of Cirrus examples.

4.1 Results of Classification.

The accuracy of the classifiers was estimated on the basis of the data from the images not used in the training. This gives us a measure of the classifiers ability to generalise and cope with unseen data. If the relative performance of the classifiers is consistent over all training/test sets then we might be confident that this is true for the techniques in general.

The results are shown in Table 1, ordered left-to-right on the basis of the Bayesian classifier. The comparative performance can be seen more clearly from the graph in Figure 7.

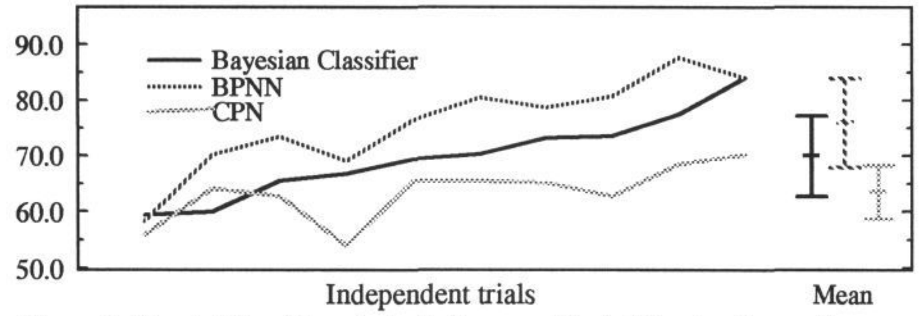


Figure 6 : The right hand lines indicate the mean ±1 sd of the classifier performance.

Classifiers	Performance on 10 independent trials										Ave.
Bayesian	59.4	59.9	65.4	66.7	69.3	70.2	73.2	73.5	77.4	84.0	69.9
BPNN	58.4	70.1	73.4	68.9	76.3	80.5	78.6	80.7	87.7	84.0	75.9
CPN	56.0	64.0	62.7	54.0	65.4	65.4	65.1	62.7	68.4	70.1	63.4

Table 1 : Comparison of Classifiers on random training and test set pairs

The BPNN is consistently much better then the CPN. This inferiority has been recognised by the CPN's creator Hecht-Nielsen [2], who advocates that since it is so simple to build, and quick to learn, it should only be used to generate rapid *prototype* NN solutions, with the final implementation using BPNN or some other architecture.

The BPNN also appears to be perform significantly better than the Bayesian approach. Out of the ten trials, nine show an improvement between 2.2 and 10.3%. In the remaining trial, which also happens to be the least successful trial, the BPNN was 58.4% correct compared to 59.4% for the Bayesian Classifier.

5 Conclusions.

The Neural Network approach has been put forward because of its independence of any *a priori* model of the class distributions. This means it can accurately model statistical distributions, for which the normal assumptions made by the Bayesian classifier are invalid. For nearly all of the independent trials the BPNN performs better than the Bayesian classifier.

An insight into the characteristics of the classifiers can be gained by examining their performance with respect to each individual class. The greatest difference in performance lies with cirrus, the BPNN performs on average 20% more accurately than either of the other classifiers. This might be explained if the Cirrus feature distributions were markedly non-normal, however it is unclear whether this is due to the nature of Cirrus features, or an *artifact* of the limited number of cirrus examples.

The performance of the BPNN is slightly worse than that of the Bayesian classifier when dealing with sky. The majority of the BPNN error is due to mislabelling sky as cirrus. This is most probably a necessary trade-off which allows BPNN to perform much better on cirrus than the other classifiers. Even disregarding the results of the classification of Cirrus, the BPNN performance is at least as good as the Bayesian.

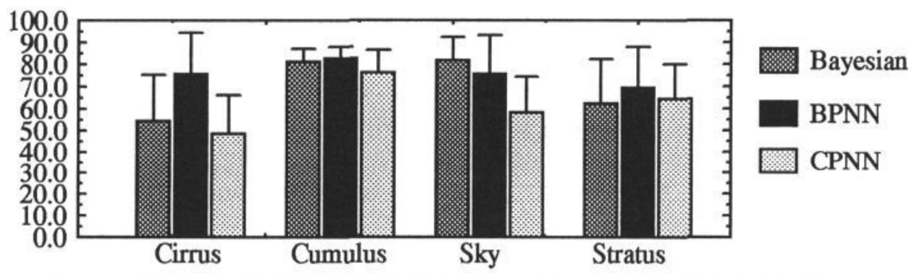


Figure 7 : The graphs represent the average performance of the classifiers (±1 sd), with respect to the individual classes.

It appears from our results that Neural Networks do offer a very useful means of classifying images without the need for a statistical model of the features, however the training parameters require careful selection. The major disadvantage of using the BPNN is the lack of clear strategy for choosing between the many training paradigms. Different problems appear to require different training mechanisms, or different parameters for the same training mechanism. Finding the best parameters, and training algorithms is a matter of experience, trials and patience.

Acknowledgements: This work was carried out with support from the Science and Engineering Research Council, under CASE award 90593345, in collaboration with the Meteorological Office. Thanks are due to H.J. Grech-Cine for many useful discussions, and practical assistance with the Neural Network software.

6 References.

[1] Bediktsson J, Swain P.H, Ersoy O.K, "Neural Network Approaches Versus Statistical Methods in Classification of Multi-source Remote Sensing Data", IEEE Transactions on Geoscience, and Remote Sensing, Vol. 28, No. 4, July 1990, pp 543-552.

[2] Hecht-Nielsen R, "Neurocomputing", Addison-Wesley Publishing Company, 1989.

[3] Hinton G.E, "Neural Networks: The 1st SUN Annual Lecture in Computer Science at the University of Manchester", July 1989, Lecture slides.

[4] Laws K.I, "Textured Images Segmentation", PhD dissertation, University of Southern California, 1980.

[5] Lee J, Weger R.C, Sengupta S.K, Welch R.M, "A Neural Network Approach to Cloud Classification", IEEE Transactions on Geoscience and Remote Sensing, Vol 28, No. 5, September 1990, pp 846-855.

[6] Richards K, Sullivan G.D, and Jones D.W, "Colour Segmentation of Cloud Image Data.", World Meteorological Organisation Technical Conference on Instruments and Methods of Observation, Vienna, 1992, pp 295-299.

[7] Unser M, "Local linear transforms for texture measurements", Signal Processing 11, 1986, pp.61-79.

[8] Visa A, Valkealahti K, Simula O, "Cloud Detection Based on Texture Segmentation by Neural Network Methods", IEEE International Joint Conference on Neural Networks, Volume 2, 1991, pp 1001-1006.

[9] Wasserman P.D, "Neural Computing: Theory and Practise", Van Nostrand Reinhold, 1989, pp 63-71.

[10] Young T.Y, King-Sun F, "Handbook of Pattern Recognition and Image Processing", Academic Press Inc., 1986, pp22-24.