Kohonen Neural Networks for Optimal Colour Quantization *

Anthony H. Dekker
dekker@ACM.org

Abstract

We present a Self-Organizing Kohonen Neural Network for quantizing colour graphics images. The network is compared with existing algorithmic methods for colour quantization. It is shown experimentally that, by adjusting a quality factor, the network can produce images of much greater quality with longer running times, or slightly better quality with shorter running times than the existing methods. This confounds the frequent observation that Kohonen Neural Networks are necessarily slow. The continuity of the colour map produced can be exploited for further image compression, or for colour palette editing.

1 Introduction

Colour computer graphics is one of the most popular applications of computer technology today. The best representation of a colour image is an array of pixels, each pixel being a triple (R, G, B) of intensities for the three primary colours red, green, and blue. Each intensity is usually represented in the range $0 \ldots 255$. This requires 3 bytes of storage per pixel, which consumes excessive space, and requires large and expensive *frame buffers* for display [6, p 342].

A solution to these problems is to *quantize* the image using a table of up to 256 distinct colours $(R_0, G_0, B_0), \ldots, (R_{255}, G_{255}, B_{255})$. Each pixel (R, G, B) is then replaced by a single byte i which indicates the most similar colour (R_i, G_i, B_i) in the table. The table (or *map*) must be chosen in such a way that when pixels (R, G, B) are replaced by (or *mapped to*) the colours (R_i, G_i, B_i) in the table, the resulting errors are as small as possible.

The pixels (R, G, B) in the input image can be viewed as points in a $256 \times 256 \times 256$ cube (the *colour space*). Figure 1 shows the distribution of 122,227 distinct points from a test image (the image itself is shown in figure 5). The lower left of the figure shows the distribution of points in the cube, with the origin $(0, 0, 0)$ at the lower left of the diagram. The blue coordinate (B) is shown on the vertical axis, red (R) on the horizontal, and green (G) extending into the page. Projections of this cube from the top, front and side are shown around it (clockwise from the top left). Shades of grey are represented by points on the main diagonal from $(0, 0, 0)$ to $(255, 255, 255)$. Most points are clustered near the main diagonal, reflecting the fact that the colours in this image are not fully saturated.

There are two basic approaches to colour quantization, which we call *pre-clustering* and *post-clustering*. Both approaches divide the pixels into 256 clusters, and choose a representative (R_i, G_i, B_i) for the i^{th} cluster. In *pre-clustering* we first divide the pixels into 256 clusters, and then choose the arithmetic mean or mode of all the pixels in the i^{th} cluster as the representative (R_i, G_i, B_i). Currently available quantization methods all use pre-clustering, with varying methods for choosing clusters. In *post-clustering* we first choose 256 representatives (R_i, G_i, B_i) and place pixels in the cluster corresponding to the representative to which they are closest. This requires a measure of the *distance* between a pixel (R, G, B) and each representative (R_i, G_i, B_i). Although Euclidean distance is the

*An edited version of this paper appeared in *Network: Computation in Neural Systems*, Volume 5, 1994, pages 351–367, Institute of Physics Publishing.

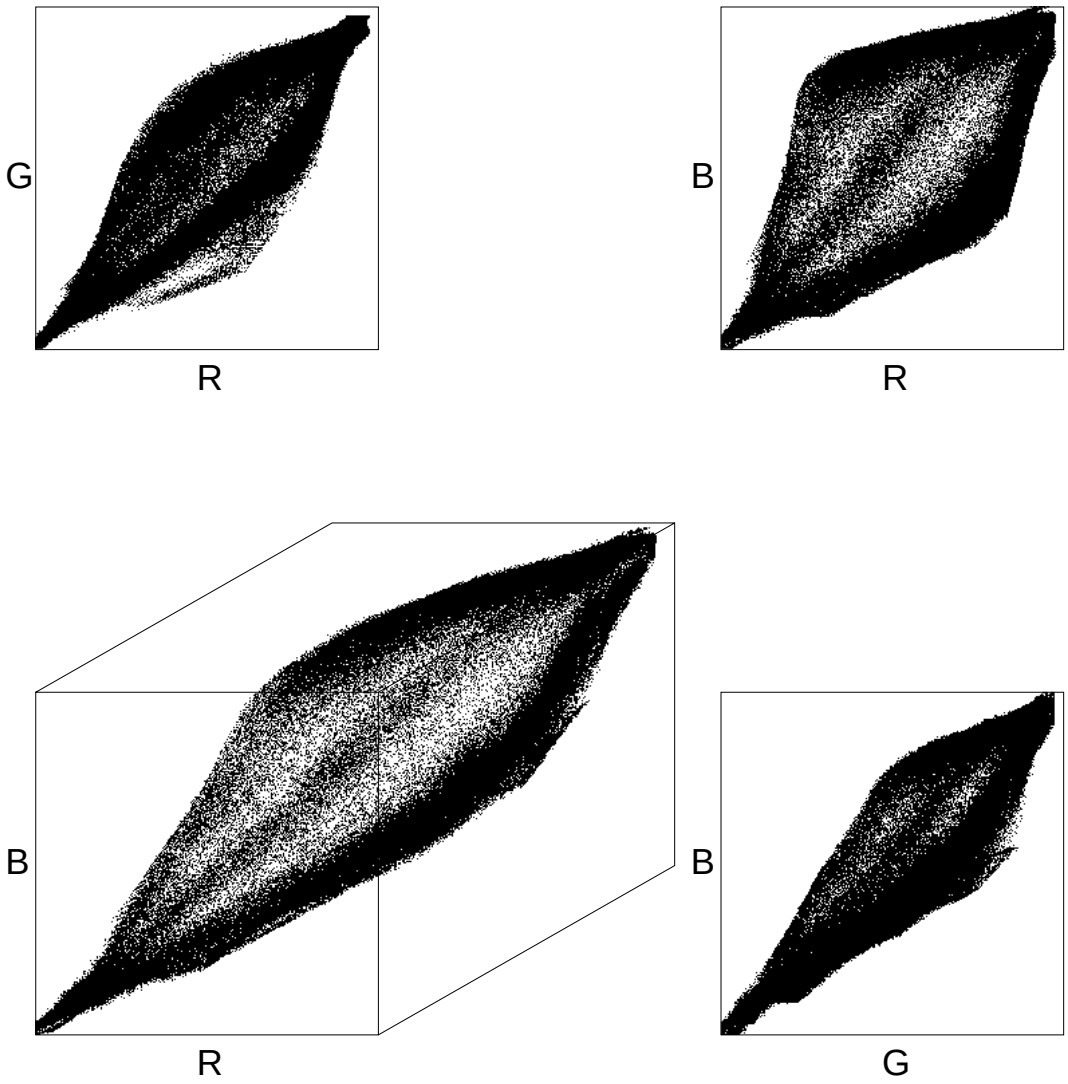

Figure 1: Colour distribution for an image with 122,227 distinct colours

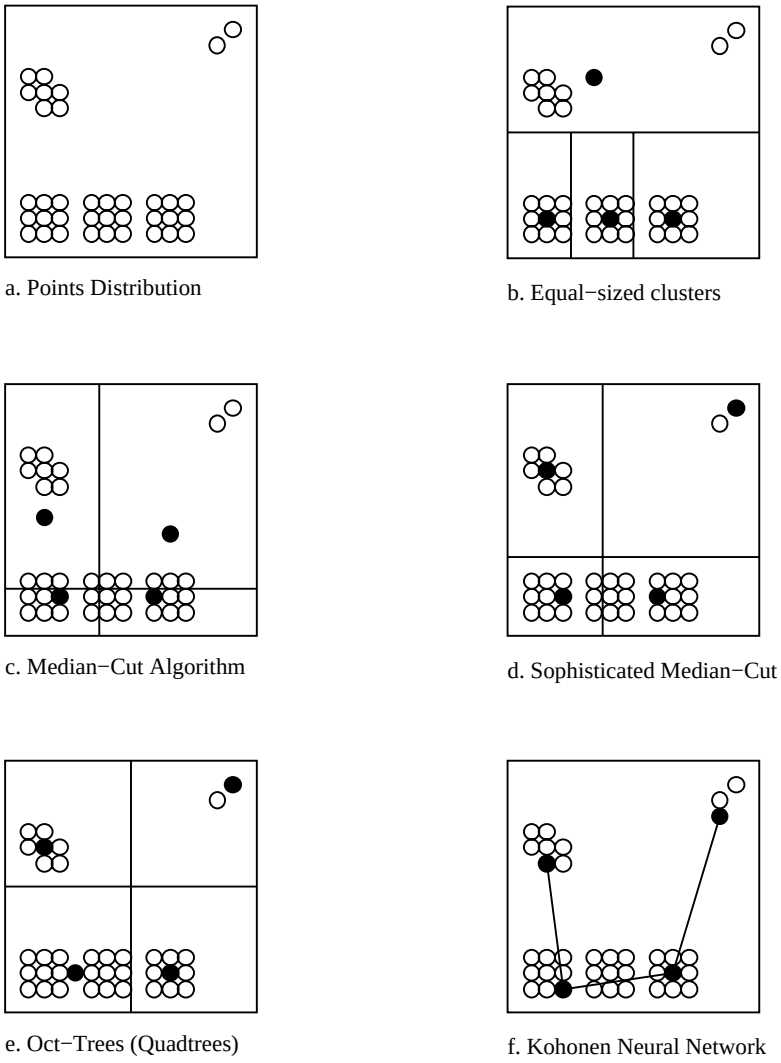

Figure 2: Colour quantization on a 16×16 square

most obvious, and similarity to the human eye is perhaps the most accurate, for ease of computation we use the 'Manhattan' distance:

$$d = |R - R_i| + |G - G_i| + |B - B_i|$$

The quantization technique we propose uses post-clustering, with the choice of representatives being made using a *Self-Organizing Neural Network* [15].

The effectiveness of a quantization technique can be evaluated most simply by finding the mean distance $|R - R_i| + |G - G_i| + |B - B_i|$ between pixels and their representatives. We call this the *mean mapping error*, and this error should be as small as possible. Although our experiments are performed on 24-bit colour images, the colour quantization process process can be illustrated more clearly by considering a 16×16 space of pairs (x, y), choosing four representatives instead of 256. Figure 2a shows a sample distribution of 36 pairs as open circles, and figure 2b-f shows the result of various quantization techniques. The mean mapping errors for each technique are shown in table 1. Representatives are shown as filled circles, and horizontal and vertical lines divide the clusters.

One approach to quantization is to attempt to form equal-sized clusters (figure 2b). However, this is not only difficult, but it does not produce the most effective result. Images often contain small groups of pixels isolated in the colour space (e.g. specular highlights). These will not be represented accurately using equal-sized clusters. One way of achieving

approximately equal-sized clusters is Heckbert's *Median-Cut* algorithm [6, p 344] which repeatedly forms histograms of pixel occurrences, and divides volumes of colour space along the median, until the required number of clusters is obtained (figure 2c). A more sophisticated version of Median-Cut will make the division to one side of the median if the pixel distribution contains a suitable 'gap' (figure 2d).

The *Oct-Tree* algorithm due to Gervautz and Purgathofer [6, p 345] repeatedly subdivides a cube into 8 smaller cubes in a tree-like fashion (for our simple example, we divide a square into 4 smaller squares). The algorithm then repeatedly combines adjacent cubes which contain the least number of pixels, until the desired number of clusters is obtained. This process may result in slightly less than 256 clusters, although it has the advantage of placing isolated groups of points in their own cluster (figure 2e).

The technique we propose uses a one-dimensional Self-organizing Neural Network [15]. The network contains one neuron for each desired cluster. Through the learning process, each neuron acquires a *weight vector* (R_i, G_i, B_i) which is used as a representative. After learning is completed, pixels are mapped to the closest weight vector. For our simple example, weight vectors are pairs, and figure 2f shows a network with 4 neurons after learning. Adjacent neurons are connected by line segments, and filled circles indicate the weight vectors, which are used as representatives. It can be seen that for this example the total length of line segments connecting adjacent neurons is the smallest possible. In general, experiments show that the average distance between neurons is kept small, although there are no theoretical results confirming this for the general case.

Table 1: Mean Mapping Errors for Simple Example

Technique		Error
b.	Equal-sized clusters	2.39
c.	Simple Median-Cut	3.53
d.	Sophisticated Median-Cut	1.86
e.	Oct-Trees (Quadtrees)	1.94
f.	Kohonen Neural Network	2.14

2 Kohonen Neural Networks

Kohonen Neural Networks [15, 18] and [14, sections 3.4 and 4.4] are a form of self-organizing neural network which define a mapping from a subset of $\mathbf{R}^n$ to $\mathbf{R}^m$, where $m \leq n$. The mapping is observed to have three important properties: it is continuous almost everywhere on its domain, the reverse map is continuous, and the output of the map provides close to the maximum possible information about the input. However, there is no general proof that these properties hold for every network. An analytical solution for the final network can be given for the case $n = m = 1$ [15, 18], or for $m = 1$ and a *neighbourhood* of fixed radius 0 or 2 [4]. For other cases, no analytical solution has been found. The theoretical properties of Kohonen Neural Networks are also discussed in [8]. These networks are based on the behaviour of topological maps in the cerebral cortex of the brain. They have been applied to areas such as speech recognition [3, chapter 5], pattern recognition [16], creativity in theorem-proving [10], the learning of ballistic movements [19], and modelling aerodynamic flow [13].

We use a Kohonen Neural Network consisting of a one-dimensional array of 256 neurons, each containing a weight vector (R_i, G_i, B_i). The network defines a mapping $\mathcal{M}$ from a triple (R, G, B) to the index i of the closest weight vector. This mapping induces a function $\mathcal{F}$ from that subset of the cube $[0, 255]^3$ which is occupied by pixels, to the interval $[0, 255]$, defined by connecting adjacent weight vectors by line segments, as is done in figure 2f. Figure 3 shows the network corresponding to figure 1, using the same format. The line segments

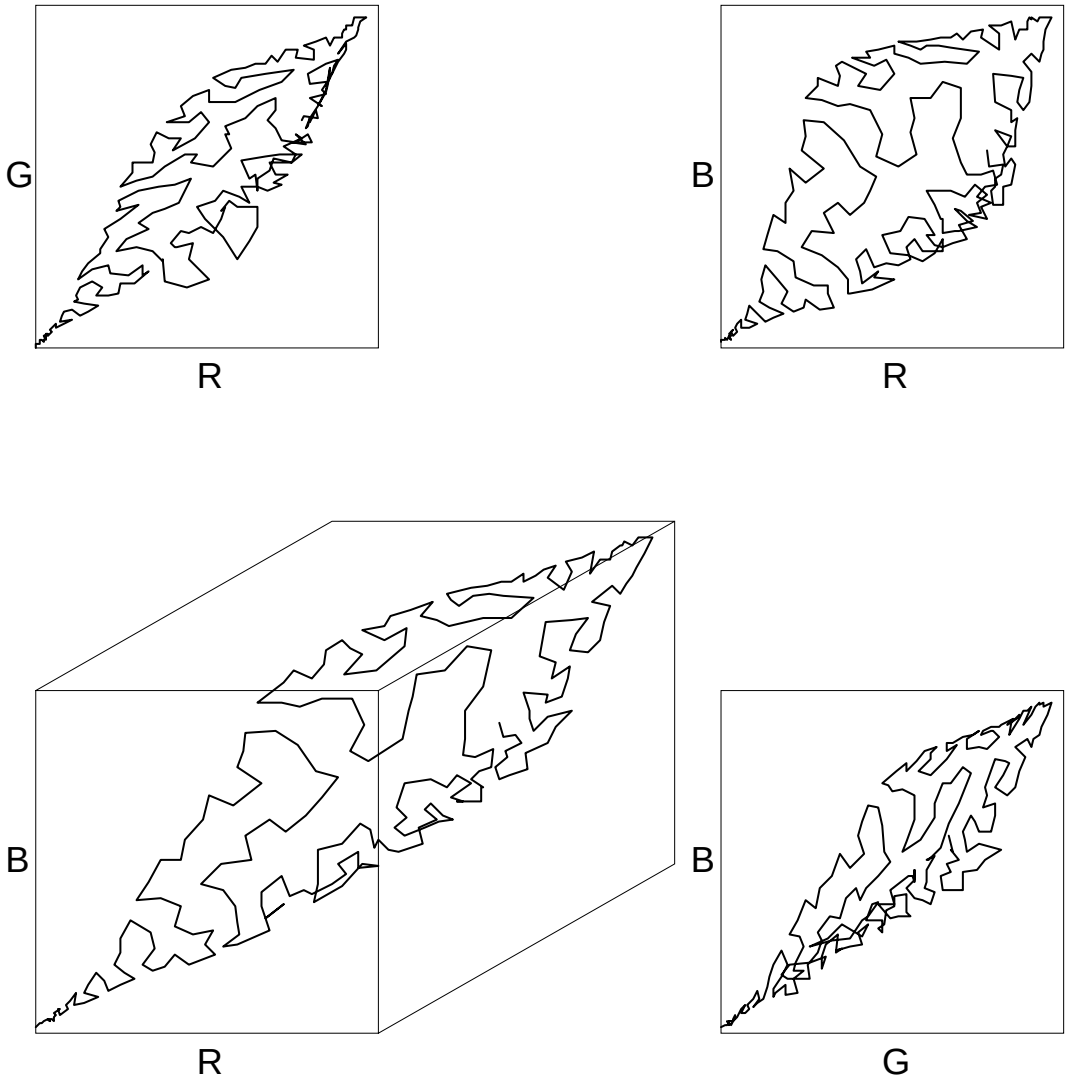

Figure 3: Kohonen Neural Network for an image with 122,227 distinct colours

in these figures form a single open curve $\mathcal{C}$ defined by:

$$\mathcal{C}_x = (1 + \lfloor x \rfloor - x)(R_{\lfloor x \rfloor}, G_{\lfloor x \rfloor}, B_{\lfloor x \rfloor}) + (x - \lfloor x \rfloor)(R_{1+\lfloor x \rfloor}, G_{1+\lfloor x \rfloor}, B_{1+\lfloor x \rfloor})$$

where x ranges from 0 to 255. The function $\mathcal{F}$ then maps each point in $[0, 255]^3$ to the closest point on the curve $\mathcal{C}$. This 'closeness' is in terms of Euclidean distance. The fact that our implementation uses 'Manhattan' distances for efficiency is not a problem, since convergence of one distance to zero implies convergence of the other. The function $\mathcal{F}$ is continuous on its domain except for a finite number of surfaces which are equidistant from two parts of the curve, i.e. they bisect 'loops' in the network. These 'discontinuity surfaces' often occur in parts of the domain thinly occupied by pixels, as is seen when figures 3 and 1 are compared. The continuity of $\mathcal{F}$ almost everywhere means that similar colours are almost always mapped to nearby indexes i. The inverse function is defined by $\mathcal{F}^{-1}(x) = \mathcal{C}_x$, and is trivially continuous. However, since the line segments in the curve tend to be relatively short, adjacent weight vectors (R_i, G_i, B_i) will usually be similar colours. Finally the mapping has the properties that the mean mapping error $|R - R_i| + |G - G_i| + |B - B_i|$ is small, and that each index i is output with approximately equal frequency (except where there are isolated groups of pixels).

The network must map an input (R, G, B) to the index i which minimises the mapping

error

$$d = |R - R_i| + |G - G_i| + |B - B_i|$$

To do this efficiently, the network (after training is completed as described below) is converted to a table of quadruples (R_i, G_i, B_i, i), sorted and indexed on the green component G_i. In searching for the minimum value of d, values of G_i close to G are examined first, and the current smallest distance $d_{\min}$ is used to limit the search to values of G_i in the range $G - d_{\min} < G_i < G + d_{\min}$. In order to avoid any consistent bias, values of G_i above and below G are examined alternately. As smaller values of d are found, the search space grows smaller. Our experiments show that on average, only 11% of the table is searched before the minimum value of d is found.

3 The Learning Algorithm

Initially we set the weight vectors with $R_i = G_i = B_i = i$. Kohonen [15, p 135] suggests initially assigning random values near the centre of the cube, but we have found that for this application, initialising the weight vectors to positions on the main diagonal is a good first approximation to the input. We then repeatedly scan input pixels (R, G, B) and find the 'best' weight vector (R_i, G_i, B_i) corresponding to the input. This vector is then updated by moving it closer to the input:

$$(R_i, G_i, B_i) := \alpha(R, G, B) + (1 - \alpha)(R_i, G_i, B_i)$$

Here α is a parameter which is initially 1, and decreases with time. Kohonen [15, p 133] suggests decreasing α linearly, but we have found that results are improved and training time is decreased if α decreases *exponentially* from 1 at the start of training (cycle 0) to 0.05 at the end of training (cycle 99), i.e. the value of α at cycle t is given by:

$$\alpha = e^{-0.03t}$$

Although this definition of α does not satisfy the guaranteed convergence criterion in [18, p 260], it performs extremely well for this application, requiring less training than is usual in Kohonen Neural Networks.

As usual, we consider the network to be slightly 'elastic' in that when a weight vector is updated, the neighbouring vectors are also moved. Specifically, there is a neighbourhood of radius r, which decreases with time, and for $i - r \le j \le i + r$ (and $0 \le j \le 255$), we update the vectors in the neighbourhood by:

$$(R_j, G_j, B_j) := \alpha\rho_{(i,j,r)}(R, G, B) + (1 - \alpha\rho_{(i,j,r)})(R_j, G_j, B_j)$$

where $\rho_{(i,j,r)}$ is equal to 1 if $i = j$, decreasing as $|i - j|$ increases, down to 0 when $|i - j| = r$. We have found by experience that, as is the case with α, the values of r used significantly affect performance. The best results for this application are obtained when r decreases exponentially from 32 at cycle 0 until r becomes less than 2 at cycle 86, i.e. the value of r at cycle t is given by:

$$r = 32e^{-0.0325t}$$

This definition of r is combined with the following definition of $\rho_{(i,j,r)}$:

$$\rho_{(i,j,r)} = 1 - \left(\frac{|j - i|}{\lfloor r \rfloor}\right)^2$$

where $\lfloor r \rfloor$ is the integer part of r. A consequence of this definition is that for $j = i \pm \lfloor r \rfloor$, (R_j, G_j, B_j) is unchanged, and hence when $r < 2$ (i.e. the last 14 cycles), only (R_i, G_i, B_i) is updated. Figure 4 shows this updating process for the case where $r = 2$ and $\alpha = 0.6666$. The network before update is shown by solid lines and crosses, and the updated network is shown by dashed lines and open circles. The closest neuron is moved two-thirds of the distance to the new data point (as indicated by the arrow) and the two neighbouring neurons are moved half the distance (since $\rho_{(i,j,r)} = 0.75$).

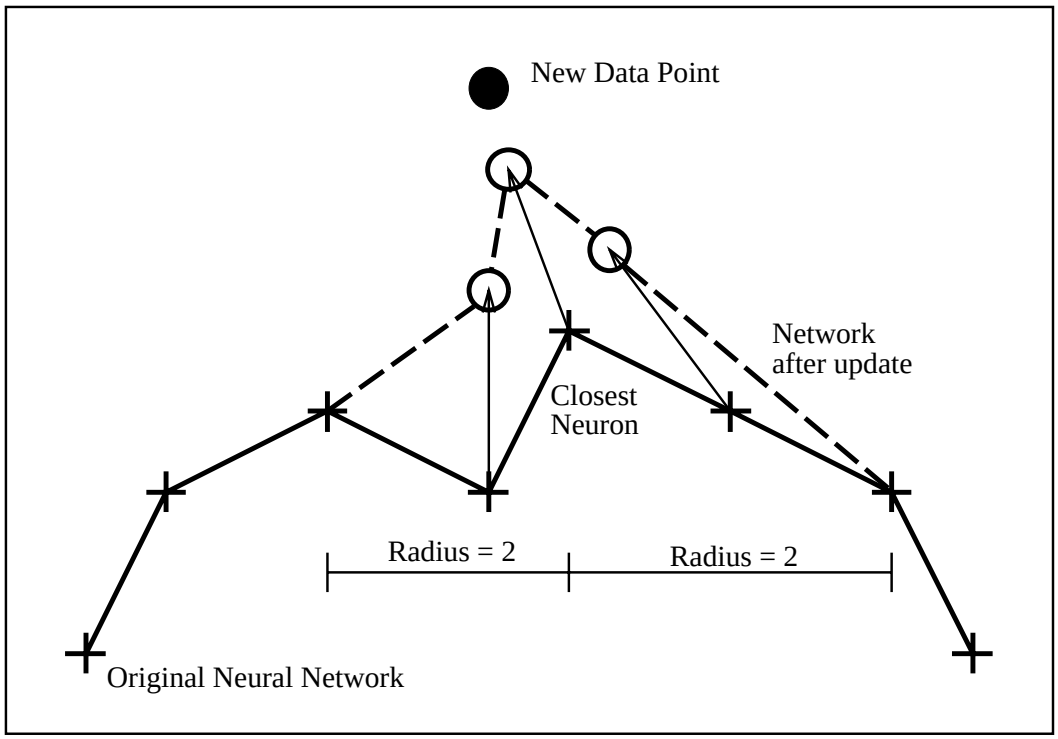

Figure 4: Kohonen Neural Network update algorithm

The input values are obtained by examining all the pixels in the input image exactly once, dividing the scan into 100 cycles (sometimes a partial 101^{st} cycle is needed). If the input image contains N pixels, $\frac{N}{100}$ pixels are examined in each cycle. This is done by choosing every P^{th} pixel, where P is a prime number close to 500 which is not a factor of N (our implementation chooses P to be 487, 491, 499 or 503). Thus each cycle involves scanning the image 5 times, and after 100 cycles when every pixel has been examined once, training stops. This multi-scan process is necessary because the learning process requires the inputs to be randomly distributed. It would be possible to examine each pixel twice, but this does not result in an improvement in performance. On the other hand, if only a subset of pixels is examined, a lower-quality quantized image is obtained, but the algorithm runs much faster.

We have not yet defined the 'best' vector corresponding to an input. Kohonen [15, p 131] suggests it be the closest vector, or the most highly correlated one. However, this does not ensure a fair assignment of weight vectors to different regions of the cube. In particular, if all pixels are located in two non-overlapping regions A and B, with weight vector 0 located in region A and weight vectors $1 \ldots 255$ in region B, then for every pixel in A, weight vector 0 will always be the closest, and the weight updating process will move weight vector 0 towards the centre of region A. After training is completed, all pixels in region A will be mapped to weight vector 0, causing considerable distortion.

The solution to this problem was discovered by Desieno [14, p 69]. The 'best' vector for the input (R, G, B) is the weight vector (R_i, G_i, B_i) minimising:

$$|R - R_i| + |G - G_i| + |B - B_i| - b_i$$

where b_i is a bias factor which increases for less frequently chosen vectors. This allows a vector not to be chosen if it has been chosen 'too many' times already. The bias b_i is defined by:

$$b_i = \gamma\left(\frac{1}{256} - f_i\right)$$

where γ is a constant, and f_i estimates the frequency at which weight vector i is closest to the input. Initially $f_i = \frac{1}{256}$ and hence $b_i = 0$. When scanning a pixel, we first find the weight vector closest to the pixel (this must be done by a linear scan, since the network is in a state of flux), and update f_i with:

$$f_i := f_i - \beta f_i + \beta$$

while the other weight vectors are updated with:

$$f_i := f_i - \beta f_i$$

Since f_i and b_i change slowly, we can perform the search for the closest weight vector to the input in the same loop as the search for the 'best' vector to be updated, rather than updating b_i before searching for the 'best' vector. We can also update b_i directly with:

$$b_i := b_i + \gamma\beta f_i - \gamma\beta$$

for the closest vector to the pixel, and:

$$b_i := b_i + \gamma\beta f_i$$

for the other vectors. For this application, the best performance is obtained when $\beta = \frac{1}{1024}$ and $\gamma = 1024$, i.e. $\gamma\beta = 1$. This value of β is about the same as that suggested in [14, p 69], but the value of γ is considerably larger because our input values range from 0 to 255 rather than 0 to 1. In order to maximise speed, we use integer arithmetic where possible.

With this scheme, if region A contains only one weight vector, that vector will develop a large f_i, and hence a large negative b_i. As a consequence, some vectors in region B will be considered to be 'best,' and will be moved to region A. This scheme therefore ensures that each neuron forms the centre of a cluster of pixels of approximately equal size. However, the balance between distance and bias factors means that a small isolated group of pixels will still be 'allocated' a weight vector, even if the resulting cluster is smaller than average. This is precisely the behaviour that we require in a colour quantization algorithm.

4 Analysis of the Network

The frequency bias factor ensures that the clusters S_i of pixels mapping to the neurons N_i contain approximately the same number of pixels. We can approximate the shape of these clusters as spheres, with the pixels being distributed within the spheres according to a probability distribution P.

The average shift in the weight vector $\mathbf{w}_i = (R_i, G_i, B_i)$ of neuron N_i in response to input pixels can be divided into two parts $\Delta\mathbf{w}_i^\circ$ and $\Delta\mathbf{w}_i^*$. Here $\Delta\mathbf{w}_i^\circ$ is the change in response to input pixels that fall within the sphere S_i, and $\Delta\mathbf{w}_i^*$ the change in response to pixels falling outside the sphere S_i. For the last 14 steps of training, when the neighbourhood radius r is less than 2, we have $\Delta\mathbf{w}_i^* = \mathbf{0}$. As described in [18, p 243], the shift $\Delta\mathbf{w}_i^\circ$ is given by:

$$\Delta\mathbf{w}_i^\circ = \int_{S_i} \alpha(\mathbf{v} - \mathbf{w}_i) P(\mathbf{v})\, d\mathbf{v}$$

where the integral is taken over input pixels $\mathbf{v} = (R, G, B)$ within the cluster S_i. As the weight vector $\mathbf{w}_i$ moves towards the centre of gravity of the cluster, this change tends towards zero.

When the weight vectors $\mathbf{w}_i$ and $\mathbf{w}_{i+1}$ of two adjacent neurons are closer than the sum of the radii of the spheres S_i and S_{i+1}, the spheres will overlap. The overlapping region is then split into two regions T_i in which the pixels are closest to $\mathbf{w}_i$ and U_{i+1} in which the pixels are closest to $\mathbf{w}_{i+1}$. The average shift $\Delta\mathbf{w}_i^\circ$ is then given by:

$$\Delta\mathbf{w}_i^\circ = \int_{S_i - U_{i+1}} \alpha(\mathbf{v} - \mathbf{w}_i) P(\mathbf{v})\, d\mathbf{v} \; + \; \int_{U_{i+1}} \alpha\left(1 - \frac{1}{\lfloor r \rfloor^2}\right)(\mathbf{v} - \mathbf{w}_i) P(\mathbf{v})\, d\mathbf{v}$$

This holds for $r \geq 1$. If $r < 1$ the neighbourhood effect is zero, and this is equivalent to the case when $r = 1$. This equation can be simplified to:

$$\Delta \mathbf{w}_i^\circ = \int_{S_i} \alpha(\mathbf{v} - \mathbf{w}_i) P(\mathbf{v})\, d\mathbf{v} \; - \; \int_{U_{i+1}} \frac{\alpha}{\lfloor r \rfloor^2} (\mathbf{v} - \mathbf{w}_i) P(\mathbf{v})\, d\mathbf{v}$$

Since the first integral tends to zero as the weight vector $\mathbf{w}_i$ moves towards the centre of gravity of the cluster S_i, this gives a net shift of $\mathbf{w}_i$ away from the overlapping cluster S_{i+1}. Thus the overlap of the spheres S_i and S_{i+1} results in a 'repulsive force' which tends to reduce the overlap.

For the first 86 steps, when the neighbourhood radius $r \geq 2$, the average shift $\Delta \mathbf{w}_i^*$ in response to input pixels in clusters $S_{i+1}, \ldots, S_{i+r-1}$ and $S_{i-1}, \ldots, S_{i-r+1}$ is given by:

$$\Delta \mathbf{w}_i^* = \sum_{1 \leq |k| < r} \int_{S_{i+k}} \alpha \left(1 - \frac{k^2}{\lfloor r \rfloor^2} \right) (\mathbf{v} - \mathbf{w}_i) P(\mathbf{v})\, d\mathbf{v}$$

Taking the approximation $\int_{S_i} \mathbf{v} P(\mathbf{v})\, d\mathbf{v} = \mathbf{w}_i$ we obtain:

$$\Delta \mathbf{w}_i^* = \sum_{1 \leq |k| < r} \alpha \left(1 - \frac{k^2}{\lfloor r \rfloor^2} \right) (\mathbf{w}_{i+k} - \mathbf{w}_i)$$

This tends to shift $\mathbf{w}_i$ towards the centre of gravity of the clusters $S_{i-r+1}, \ldots, S_{i+r-1}$, giving the network its 'elastic' nature. However, this is balanced by the fact that neurons must be evenly distributed over the entire region of the colour space which is occupied by pixels.

It is instructive to consider the simple case where the neurons $N_{i-r+1}, \ldots, N_{i+r-1}$ are arranged in a straight line, spaced equally except for a larger space between N_i and N_{i-1}. In other words, there are vectors $\mathbf{q}$ and $\mathbf{t}$ such that for $k > 0$,

$$\begin{aligned} \mathbf{w}_{i+k} &= \mathbf{w}_i + k\mathbf{q} \\ \mathbf{w}_{i-k} &= \mathbf{w}_i - k\mathbf{q} - \mathbf{t} \end{aligned}$$

For this case we obtain:

$$\begin{aligned} \Delta \mathbf{w}_i^* &= \sum_{1 \leq k < r} \alpha \left(1 - \frac{k^2}{\lfloor r \rfloor^2} \right) k\mathbf{q} \; - \; \sum_{1 \leq k < r} \alpha \left(1 - \frac{k^2}{\lfloor r \rfloor^2} \right) (k\mathbf{q} + \mathbf{t}) \\ &= -\alpha \sum_{1 \leq k < r} \left(1 - \frac{k^2}{\lfloor r \rfloor^2} \right) \mathbf{t} \\ &= -\alpha \left(\frac{2\lfloor r \rfloor}{3} - \frac{1}{2} - \frac{1}{6\lfloor r \rfloor} \right) \mathbf{t} \\ &= -c\,\mathbf{t} \end{aligned}$$

where c is some positive number (since $r \geq 2$). Hence we have an average shift of $\mathbf{w}_i$ towards $\mathbf{w}_{i-1}$. Thus there is an 'attractive force' which tends to reduce unusually large distances between adjacent neurons. This tends to ensure that adjacent weight vectors represent similar colours, and that the network has the required continuity properties.

The compromise between the distribution of neurons over the colour space and the 'elastic' nature of the network not only gives these continuity properties, but results in the network forming a fractal space-filling curve [11], as shown in figure 3. The network in figure 3 has an estimated fractal dimension of 1.6 (the average for the twelve test images is 1.51). The exact relationship between this fractal dimension and the informational properties of the network has yet to be clarified.

5 Testing the Colour Maps

It is possible to combine the colour map produced by the Kohonen Neural Network with a Floyd-Steinberg dithering step [6, p 139] which distributes the mapping error

$$|R - R_i| + |G - G_i| + |B - B_i|$$

for a pixel to the surrounding pixels. The effect is to ensure that errors in adjacent pixels compensate for each other to some extent. However, we have found that this dithering process actually degrades image quality, since the mean mapping error produced by the Kohonen Neural Network is quite low.

We have tested the Kohonen Neural Network quantization algorithm and compared it with three existing quantization algorithms provided as part of the *xv* image processing package [5]. The *xvquick* algorithm uses post-clustering with a fixed map of 216 colours, and dithering to allow adjacent pixel errors to compensate for each other. The *xvslow* algorithm uses Median Cut quantization together with dithering, while the *xvbest* algorithm, originally due to [17], uses Sophisticated Median Cut without dithering. We have also compared the Kohonen Neural Network quantization algorithm with a non-dithering implementation of the Oct-Tree algorithm.

We provide two comparisons of algorithm performance. The first is simply the mean mapping error $|R - R_i| + |G - G_i| + |B - B_i|$, while the second is the mean mapping error calculated after a *smoothing* operation on both the input and output images, using the following filter:

1	2	1
2	4	2
1	2	1

Calculating the mean mapping error after smoothing effectively calculates the error for 2×2 squares centred on each pixel. Taking into account neighbouring pixels in this way indicates the extent to which neighbouring errors cancel out, so that dithering will reduce the smoothed error while increasing the unsmoothed mean mapping error.

The following results are the mean for 12 input images (including 9 scanned photographs, 2 scanned works of art, and 1 ray-traced artificial image). All images were obtained from JPEG and GIF images available on the Internet. These images were re-sampled at half the original spatial resolution in order to restore full 24-bit colour depth. The final images ranged in size from 37,762 to 325,080 pixels with between 13,165 and 193,529 distinct colour in each image. The quantization results are summarised in table 2. For each test image, the Kohonen Neural Network gave a smaller mean mapping error than the other four algorithms. In 9 out of the 12 images the smoothed error was also smaller, in spite of the fact that no dithering was used.

Table 2: Average Mean Mapping Errors for 12 Sample Images

Algorithm	Mean Mapping Error	Smoothed Error
xvquick (dithered)	49.58	7.72
xvslow (dithered)	12.15	3.27
xvbest (non-dithered)	9.28	7.20
Oct-Trees (non-dithered)	8.96	5.92
Kohonen Network	5.34	2.96

Subjectively, the *xvquick* and *xvslow* algorithms produced images which often had visible 'grain' due to dithering. This was especially noticeable on enlargement. In the case of one complex image the *xvslow* algorithm also produced visible artifacts (patches of incorrect colour). The *xvbest* algorithm in several cases produced significant banding artifacts on smooth colour gradients (this was especially noticeable in the ray-traced image). The Oct-Tree algorithm produced images inferior in appearance to those produced by *xvbest*, with significant banding artifacts and patches of incorrect colour. Artifacts can occur with the Oct-Tree algorithm because the clustering is insufficiently flexible. In particular, when a group of pixels in the colour space is located near the intersection of several cubes, similar pixels on opposite sides of the cube boundary are quantized in different ways.

The image produced by our algorithm was similar or superior in appearance when compared with the other algorithms in each case, and showed fine detail with less distortion.

In particular, the random element in our algorithm ensures that banding artifacts are not produced, but that colour gradients appear smooth, while at the same time not producing visible 'grain.'

The CPU time taken for each algorithm on a SUN SPARCstation IPC with 24 MB of memory and a 64 kB write-through cache is summarised in table 3. Since the time depends on image size, it is reported as microseconds per pixel. This CPU time grossly underestimates the actual running time of the Oct-Tree algorithm on very large images, which can be up to 10 times longer, due to memory swapping.

Table 3: Average CPU time taken for 12 Sample Images

Algorithm	CPU time (μs/pixel)
xvquick (dithered)	17 ($\pm$ 18%)
xvslow (dithered)	72 ($\pm$ 132%)
xvbest (non-dithered)	420 ($\pm$ 201%)
Oct-Trees (non-dithered)	141 ($\pm$ 145%)
Kohonen Network	875 ($\pm$ 6%)

It can be seen that our basic algorithm is on average twice as slow as *xvbest*, 6 times slower than Oct-Trees, and 12 times slower than *xvslow*. There was less variation in time with our algorithm than with the other algorithms, so that in fact our algorithm ranged from 5 times slower than *xvbest* to 40% faster.

The space overhead for our algorithm is very small: only 8 kB are needed to store the necessary arrays, in addition to the space required to store the input image. The space usage for the *xvbest* and *xvslow* quantization algorithms (excluding space to store the input image) was measured by subtracting the *xvquick* program space usage from the total *xvbest* and *xvslow* program space usage. The results are shown in table 4. The space used by *xvbest* depended on the number of distinct colours, with the line of best fit being approximately 1200 kB of fixed overhead, plus 5 kB for each 1000 distinct colours. The space used by the Oct-Tree algorithm also depended on the number of distinct colours, with the line of best fit being approximately 100 kB per 1000 distinct colours. This usage was particularly high, and led to a large amount of memory swapping. There are implementations of the Oct-Tree algorithm which reduce space usage by not building the entire tree, but these have even lower-quality output images.

Table 4: Average space usage for quantization algorithms

Algorithm	Space usage (kB)
xvslow (dithered)	521
xvbest (non-dithered)	1,100–2,100
Oct-Trees (non-dithered)	2,300–17,000
Kohonen Network	8

Following these experiments, we modified our algorithm to sample only a subset of pixels during training. Table 5 shows the performance of the algorithm for various *sampling factors* n (i.e. when sampling only $\frac{1}{n}$ of the pixels). The timing results satisfy closely the line of best fit $36 + \frac{838}{n}$ microseconds per pixel. A sampling factor of 3 gives a significant speed improvement with very little observable change in quality, running faster than *xvbest*. A sampling factor of 10 gives a slight quality reduction, but still visibly better (and with smaller mean mapping error) than existing methods, while running faster than *xvbest* or Oct-Trees. A sampling factor of 30 produces images of only slightly better quality than existing methods, but runs faster than all but *xvquick* (which uses a fixed colour map). Sampling factors greater than 30 produce only small speed improvements, and are not worthwhile. It should be noted that with sampling factors greater than 3, the smoothed error is greater

than for *xvslow* (although the mean mapping error is much lower). However, the image produced by *xvslow* is subjectively poorer than our algorithm even with a sampling factor of 30, due to the 'grain' introduced by dithering.

Table 5: Network results for various sampling factors

Factor	CPU time (μs/pixel)	Mean Mapping Error	Smoothed Error	Final α
1	875 ($\pm$ 6%)	5.34	2.96	0.05
3	314 ($\pm$ 8%)	5.55	3.27	0.05
6	175 ($\pm$ 9%)	5.81	3.54	0.05
10	119 ($\pm$ 15%)	5.97	3.71	0.06
30	65 ($\pm$ 19%)	6.53	4.26	0.10
60	52 ($\pm$ 26%)	6.73	4.45	0.15

To compensate for the reduced sampling of pixels, the value of α must be decreased more slowly than in the basic algorithm. Best performance was obtained if the final values of α were as shown in table 5.

In summary, with a useful range of sampling factors from 3 to 30, our quantization algorithm using Kohonen Neural Networks gives significantly better output images than Median Cut or Oct-Trees, or slightly better output images in less time. In addition, much less space is used. The low space overhead suggests that the algorithm would run significantly faster on a machine with a *copy-back* cache [20], where the network could be stored entirely in-cache. This would allow the weight vector updating process to be performed without memory references, speeding learning. Analysis of the machine code produced by the *cc* optimising compiler suggests that this should approximately double the speed of our algorithm, when sampling the entire image. It is somewhat surprising that sampling only $\frac{1}{30}$ to $\frac{1}{3}$ of an image can produce a good colour map, but this reflects the fact that even small features of an image contain several pixels, and the fact that our sampling (using prime number increments) is essentially random.

6 Exploiting the Topological Properties

A quantized colour image consists of a colour map $(R_0, G_0, B_0), \ldots, (R_{255}, G_{255}, B_{255})$, together with an array of one-byte pixels. Each byte $i \in 0 \ldots 255$ indexes a representative (R_i, G_i, B_i). However, if we ignore the colour map, the array can be viewed as a grey-scale image, with each byte i ranging from 0 (black) to 255 (white). We call this a *false-grey* image, since the representation is just the reverse of a *false-colour* image (which uses colour to represent grey-scale information). For most colour quantization schemes, the false-grey image appears to consist of random noise, since there is no relationship between adjacent bytes i and $i+1$. However, for our algorithm, the continuity properties discussed above ensure that an area of similar colour is usually quantized as similar bytes, and that similar bytes represent similar colours. This ensures that the false-grey image is a meaningful image. Each shade of grey in the false-grey image represents a colour in the original image.

Figure 5 shows the test image corresponding to figures 1 and 3. The image is of a woman in a faded blue denim jacket. An examination of figure 1 shows that the greatest concentrations of pixels are at the lower left of the cube (black), the upper right (white), a curve from black to white on the red side of the main diagonal (corresponding to shades of tan for the model's skin), and a curve from black to white on the blue side of the main diagonal (corresponding to shades of unsaturated blue for the denim jacket). The corresponding false-grey image is shown in figure 6.

Figure 3 shows the neural network after training. The mean mapping error for this image is 6.99. This error is kept low since the network follows the distribution of pixels, as can be seen by comparing figures 1 and 3. For discussion purposes, we can divide the network into 11 equal-sized regions $0 \ldots 10$. The network ranges from black (lower left corner, 0), to dark brown (loop towards right, 1), to dark shades of blue (large loop upwards, 2), to shades of

Figure 5: A colour image with 122,227 distinct colours

tan of increasing brightness (jagged line towards right, 3–5), to lighter shades of blue (loop at top of cube, 6–7), to white (top right, 8), to shades of tan of decreasing brightness (jagged line towards left, 9–10). The range 3–5 contains two regions of red colour, which are most noticeable on the G-R projection in the top left of figure 3. The transitions between blue and tan correspond to shades of grey (which occur in the shadowed areas of the background).

The corresponding false-grey image (figure 6) shows the regions $0 \ldots 10$ as shades of grey from black (0) to white (10). It can be seen that there are two blue regions corresponding to the jacket (the darker 2 and the lighter 6–7) and two tan regions corresponding to the model's skin (the darker 3–5 and the lighter 9–10). The second of these, which corresponds to the highlights on the model's skin, represents shades of tan of decreasing brightness. Hence in the false-grey image the lighter highlights appear darker (9) and the darker highlights appear lighter (10). This results in a visible discontinuity on the model's skin, between two shades of tan (5 and 10). Nevertheless, since there are only two major discontinuities, the false-grey image is still a meaningful image, and colour information can be deduced from it.

Since the false-grey image is a meaningful grey-scale image, it can be compressed using grey-scale image compression techniques. The fact that adjacent neurons represent similar colours ensures that small compression errors result in small changes in colour. We can represent a compressed quantized colour image by adding a 4-byte header and 768-byte colour map to the compressed false-grey image. We have compressed our test images in this way, using the JPEG compression algorithm [21, 12]. A 95% quality factor was used. For comparison, the original 24-bit image was compressed using the colour version of the JPEG algorithm. Table 6 shows the mean results for the 12 test images. The mean mapping error and smoothed error for the compressed false-grey image is shown with respect to the quantized colour image, and do not include the quantization error. For the purpose of display on an 8-bit colour display, quantization error would need to be added to both cases. The size is shown as a percentage of the original 24-bit colour image in each case.

Table 6: Comparison of Colour and False-Grey Compression

	Compressed Quantized Image	Compressed Original Colour Image
Size (% of original image)	17.1	14.4
Mean Mapping Error	10.85	7.62
Smoothed Error	5.07	3.62

Subjectively the compressed false-grey image is inferior to the compressed 24-bit colour image, with visible 'grain' similar to that produced by dithering. This is due to random high-frequency components introduced by variation between neighbours in the neural network. The compressed false-grey image is larger as well as being poorer in quality. However, the experiment does demonstrate that the false-grey image is sufficiently meaningful for grey-scale image compression algorithms to apply, with reasonable compression ratios and errors. This is not the case for grey-scale images consisting of random noise. It is anticipated that compression of the false-grey image will be suitable for applying Fractal Image Compression Techniques [1, 2] to colour images.

The fact that adjacent representatives represent similar colours also makes the colour map more meaningful for use with image processing software. Existing quantization algorithms produce a colour map which appears to be a random arrangement of 256 colours, while our algorithm produces a consistent palette of colours. In addition, colour editing of an image could usefully be performed interactively by selecting parts of the network with a mouse and 'dragging' the neural network through the colour space. The movement would follow the weight vector updating process shown in figure 4, with a neighbourhood size specified by the user. For the purpose of such colour palette editing, it would be convenient to represent the colour space in the HSV (Hue, Saturation, Value) format [6, p 333], rather than as an RGB cube.

We are currently investigating the use of Kohonen Neural Networks for predictive image

Figure 6: False-grey image for an image with 122,227 distinct colours

compression, by adding output facilities to neurons as described in [19] and [18].

7 Related Work

A preliminary version of these results was presented in [9]. Self-Organizing Neural Networks have also been independently applied to colour quantization by Chen, Kothari and Klinkhachorn [7]. They use a less sophisticated training process based on [15], but with neighbourhoods having a fixed radius of 2. However, they reduce training time by a factor of 9 by using the average of 3×3 blocks of pixels as training vectors, rather than the pixels themselves. This provides a useful way of providing lower-quality output in less time. However, since these workers do not present experimental results, it is not clear to what extent the fixed-size neighbourhood and pixel averaging degrade performance. Similar work has also been performed by Jeanny Herault at TIRF, Grenoble. However, none of the previous approaches has been widely adopted for practical use.

8 Conclusion

We have presented a new colour quantization algorithm for mapping 24-bit colour images to 8-bit colour, using self-organizing Kohonen Neural Networks. With limited sampling, our algorithm can produce output which is slightly better than that of the Oct-Tree and Median-Cut algorithms, while running more quickly. If more pixels are sampled (at a significant speed penalty), very high quality images are obtained, with almost half the mean mapping error, and with no contouring or other artifacts. The algorithm uses very little space (only 8 kB in addition to the space used to store the input image).

The colour map produced by our algorithm has useful continuity properties: similar colours are usually quantized to similar representatives, and adjacent representatives represent similar colours. These properties allow the quantized image to be compressed using grey-scale compression techniques, although the result is inferior to colour JPEG compression. The continuity properties also make the colour map more meaningful for use with image processing software.

9 Acknowledgements

The author is indebted to Pushkar Piggott, two anonymous referees, and several email correspondents for useful comments on Kohonen Neural Networks and on drafts of this paper.

References

[1] Michael Barnsley. *Fractals Everywhere.* Academic Press, 1988.

[2] Michael F. Barnsley and Lyman P. Hurd. *Fractal Image Compression.* AK Peters Ltd., Wellesley, Massachusetts, 1993.

[3] R. Beale and T. Jackson. *Neural Computing: An Introduction.* Adam Hilger, Bristol, 1990.

[4] Catherine Bouton and Gilles Pagès. Self-organization and a.s. convergence of the one-dimensional Kohonen algorithm with non-uniformly distributed stimuli. *Stochastic Processes and their Applications*, 47(2):249–274, September 1993.

[5] John Bradley. *xv - interactive image display for the X Window System.* University of Pennsylvania, Feb 1992. On-line UNIX Manual.

[6] Peter Burger and Duncan Gillies. *Interactive Computer Graphics: Functional, Procedural, and Device-Level Methods.* Addison-Wesley, 1989.

[7] X. Chen, R. Kothari, and P. Klinkhachorn. Reduced color image based on adaptive palette color selection using neural networks. In *World Congress on Neural Networks, Portland, Oregon, 11–15 July 1993*, volume I, pages 555–558. INNS Press, 1993.

[8] Marie Cottrell and Jean-Claude Fort. Etude d'un algorithme d'auto-organisation. *Ann. Inst. Henri Poincaré*, 23(1):1–20, 1987.

[9] Anthony Dekker. Optimal colour quantization using kohonen neural networks. Technical Report TR10/93, Department of Information Systems and Computer Science, National University of Singapore, October 1993.

[10] Anthony Dekker and Paul Farrow. Creativity, chaos, and artificial intelligence. In *Proceedings of the Symposium on AI, Reasoning & Creativity, Lamington National Park, Queensland, Australia, 20–23 August 1991.* Kluwer, to appear.

[11] P. Grassberger. Estimating the fractal dimensions and entropies of strange attractors. In Arun V. Holden, editor, *Chaos*, pages 291–311. Manchester University Press, 1986.

[12] Independent JPEG Group. *cjpeg - compress an image file to a JPEG file*, Feb 1992. On-line UNIX Manual.

[13] Robert Hecht-Nielsen. Neurocomputing: picking the human brain. *IEEE Spectrum*, 25(3):36–41, March 1988.

[14] Robert Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, 1990.

[15] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, third edition, 1989.

[16] Teuvo Kohonen. Pattern recognition by the self-organizing map. In Eduardo R. Caianiello, editor, *Parallel Architectures and Neural Networks: Third Italian Workshop, Vietri sul Mare, Salerno, 15–18 May, 1990*, pages 13–18. World Scientific, Singapore, 1990.

[17] Jef Poskanzer. *ppmquant - quantize the colors in a portable pixmap down to a specified number*, Jan 1991. On-line UNIX Manual.

[18] Helge Ritter, Thomas Martinetz, and Klaus Schulten. *Neural Computation and Self-Organizing Maps: An Introduction.* Addison-Wesley, 1992.

[19] Helge Ritter and Klaus Schulten. Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements. In Rolf Eckmiller and Christoph v.d. Malsburg, editors, *Neural Computers*, pages 393–406. Springer-Verlag, Berlin, 1989.

[20] A. J. van de Goor. *Computer Architecture and Design.* Addison-Wesley, 1989.

[21] Gregory K. Wallace. The JPEG still picture compression standard. *CACM*, 34(4):30–44, April 1991.