Optimal Colour Quantization using Kohonen Neural Networks

Anthony H. Dekker

Department of Information Systems and Computer Science
National University of Singapore
Kent Ridge, Singapore 0511
e-mail: `tdekker@iscs.nus.sg`

October 8, 1993

Abstract

A colour quantization technique is presented which maps 24-bit colour images to 8-bit colour, using Self-Organizing Neural Networks. The quantized image is superior to that produced by existing methods, and has useful continuity properties.

1 Introduction

Colour computer graphics is one of the most popular applications of computer technology today. A colour image can be represented—as is done on a TV screen—using an array of pixels, each pixel being a triple (R, G, B) of intensities for the three primary colours red, green, and blue. If each intensity is represented in the range $0 \ldots 255$, this requires 3 bytes of storage per pixel. The output of image scanners or rendering software is generally in this format.

There are, however, two problems with this representation of colour images. First, display technology requires a fast *frame buffer* to store every pixel on the screen, so that a 1024×1024 pixel screen requires a 3 MB frame buffer. This is generally very expensive. A similar objection applies to the disk space used in storing images. Second, using 3 bytes per pixel allows 16 million distinct colours to be represented, while the human eye can only distinguish about 350,000 colours, most of which will not occur in a given image. Thus the use of 3 bytes per pixel is excessive.

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. With this representation, a 1024×1024 pixel image now requires only 1.0007 MB of space. The most difficult aspect of colour quantization is the process of selecting a table (or *map*) of

256 colours for a particular image, 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. That is to say, the 256 chosen colours in the table must accurately reflect the distribution of colours in the image.

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 shall 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. All the pixels in the i^{th} cluster are mapped to this representative. 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 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* [11].

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 in a simpler form: instead of a colour space of triples (R, G, B) in a $256 \times 256 \times 256$ cube, we consider pairs (x, y) in a 16×16 square, 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. Representatives are shown as filled circles, and horizontal and vertical lines divide the clusters. We consider existing pre-clustering techniques first.

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 [5, p 344] which repeatedly forms histograms of pixel occurrences, and divides volumes (in

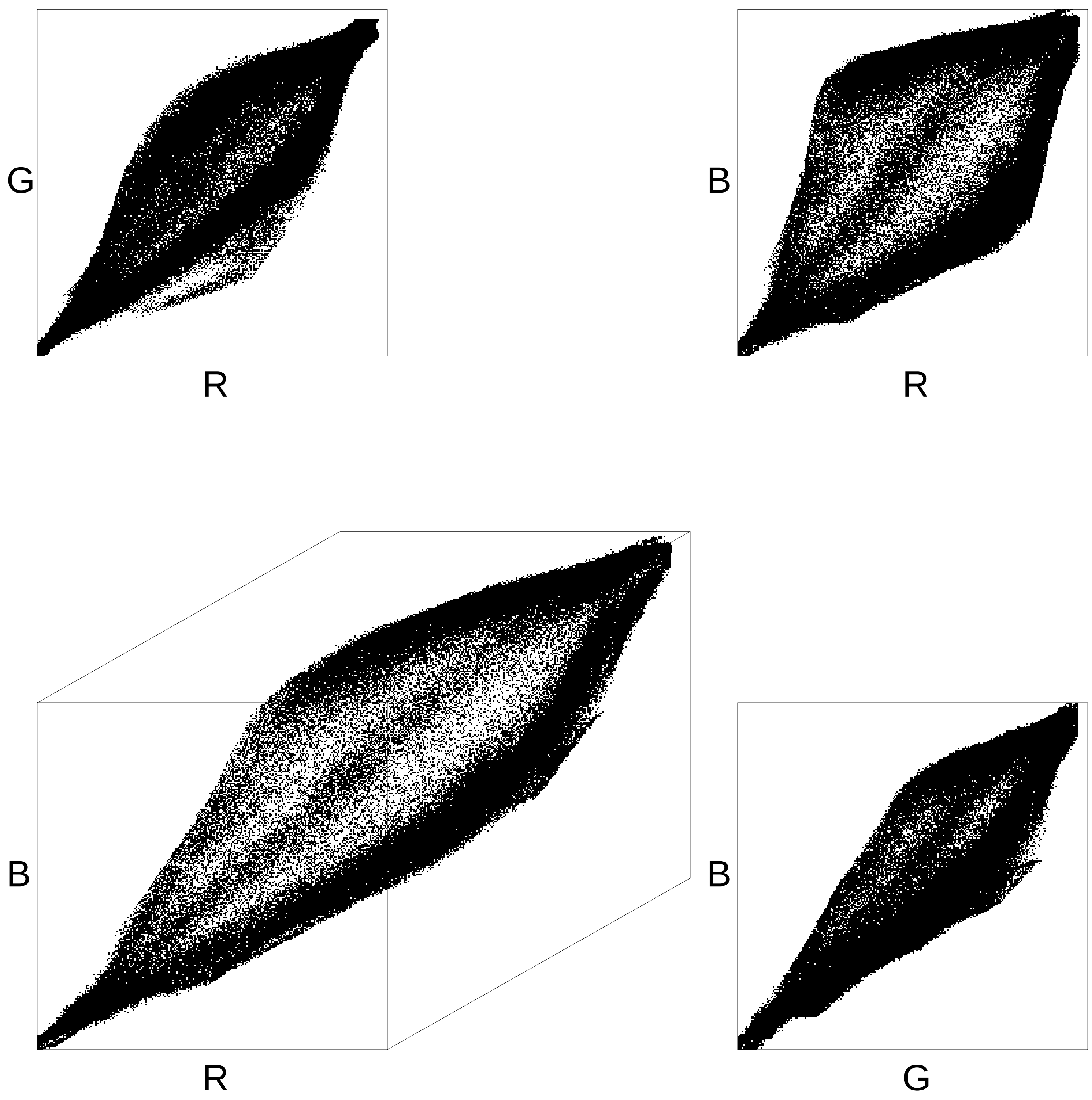
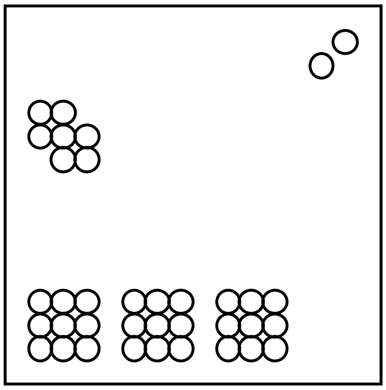
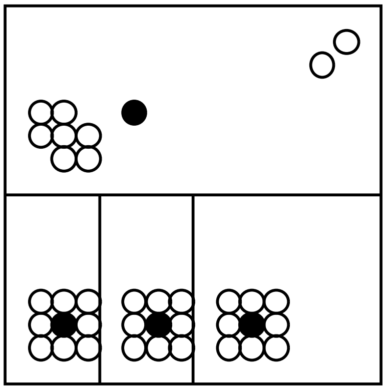


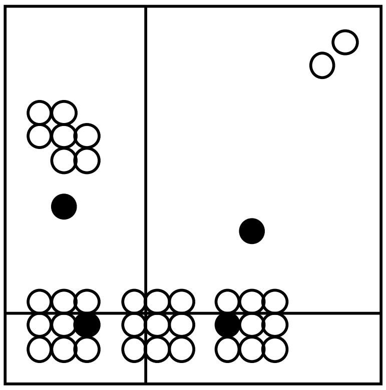
Figure 1: Colour distribution for an image with 122,227 distinct colours



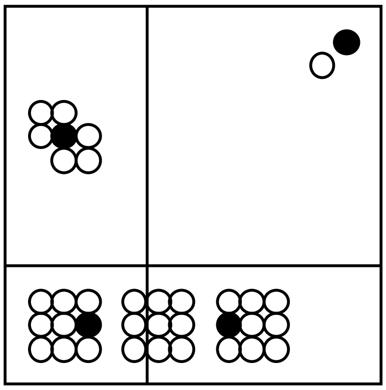
a. Points Distribution



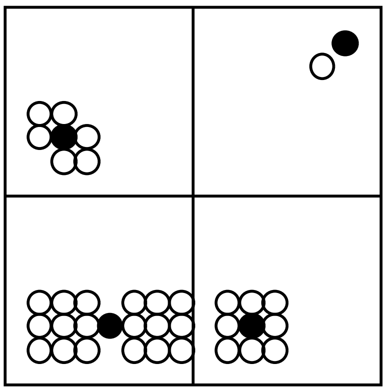
b. Equal−sized clusters



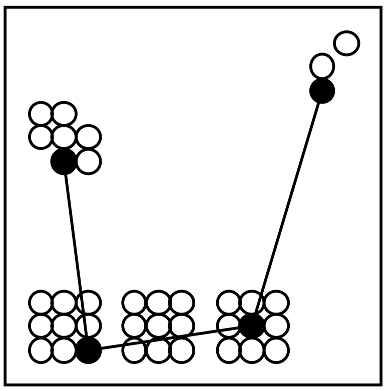
c. Median−Cut Algorithm



d. Sophisticated Median−Cut



e. Oct−Trees (Quadtrees)



f. Kohonen Neural Network

Figure 2: Colour quantization on a 16×16 square

this case, areas) 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.' For our example, this involves moving the horizontal 'cut' upwards, and gives an optimal choice of representatives (Figure 2d).

The *Oct-Tree* algorithm repeatedly subdivides a cube into 8 smaller cubes in a tree-like fashion (for our simple example, we use quadtrees, dividing a square into 4 smaller squares). The algorithm then repeatedly combines adjacent cubes (squares) which contain the least number of pixels, until the desired number of clusters is obtained. In fact, this process may result in slightly less than 256 clusters (although for the simple example we have 4 clusters). The Oct-Tree algorithm has the advantage of placing isolated groups of points in their own cluster. Figure 2e shows the subdivision for our simple example. Notice that the isolated group of 2 points forms one cluster, while a group of 18 points close to each other also forms one cluster. The Oct-Tree algorithm is generally considered to perform as well as Median-Cut [5, p 345].

The technique we propose uses a one-dimensional Self-organizing Neural Network [11]. 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 possible 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 the total length of line segments connecting adjacent neurons is the smallest possible, i.e. the average distance between neurons is kept small.

The mean mapping errors for each of the five schemes is shown in Table 1. It can be seen from this simple example that our technique has the potential to perform as well as or perhaps even better than the current best methods, Oct-Trees and sophisticated Median-Cut. Later we will give experimental results which show that our technique performs better than sophisticated Median-Cut on colour graphics images. No implementation of the Oct-Tree algorithm was available for comparison.

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

Table 1: Mean Mapping Errors for Simple Example

2 Kohonen Neural Networks

Kohonen Neural Networks [11, 14] and [10, 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 has 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. The theoretical properties of these networks are studied in [11] and [14]. An analytical solution for the final network can be given for the case $n = m = 1$. For other values of n and m, no analytical solution is possible, but the properties have been observed to hold. Kohonen Neural 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 [12], creativity in theorem-proving [6], the learning of ballistic movements [15], and modelling aerodynamic flow [9].

We use a Kohonen Neural Network to define a mapping $\mathcal{M}$ from a triple (R, G, B) to the index i of the closest representative (R_i, G_i, B_i), where $0 \leq i \leq 255$. 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 representatives (R_i, G_i, B_i) by line segments, as is done in Figure 2f. Figure 3 shows the network corresponding to Figure 1 (using the same format) and the function $\mathcal{F}$ in this case is defined on that subset of the cube shown occupied by pixels in Figure 1. The line segments 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 0 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 representatives (R_i, G_i, B_i) will usually be similar colours. Finally the mapping has the property that the mean mapping error $|R - R_i| + |G - G_i| + |B - B_i|$ is small.

The required Kohonen Neural Network consists of a one-dimensional array of 256 neurons, each containing a weight vector which is a triple of real numbers (R_i, G_i, B_i) where $0 \leq i \leq 255$. Given an input (R, G, B), the network outputs the index i which minimises the mapping error $|R - R_i| + |G - G_i| + |B - B_i|$. This is done using a loop through all weight vectors. The weight vectors are set by the learning process described below. This is done in such a way that the mean mapping error is small, and that each index i is output with approximately equal frequency. Consequently, after training is completed, each weight vector is surrounded by a cluster of pixels which are closer to that vector than any other, and the weight vector acts as the representative for the cluster. We have seen that

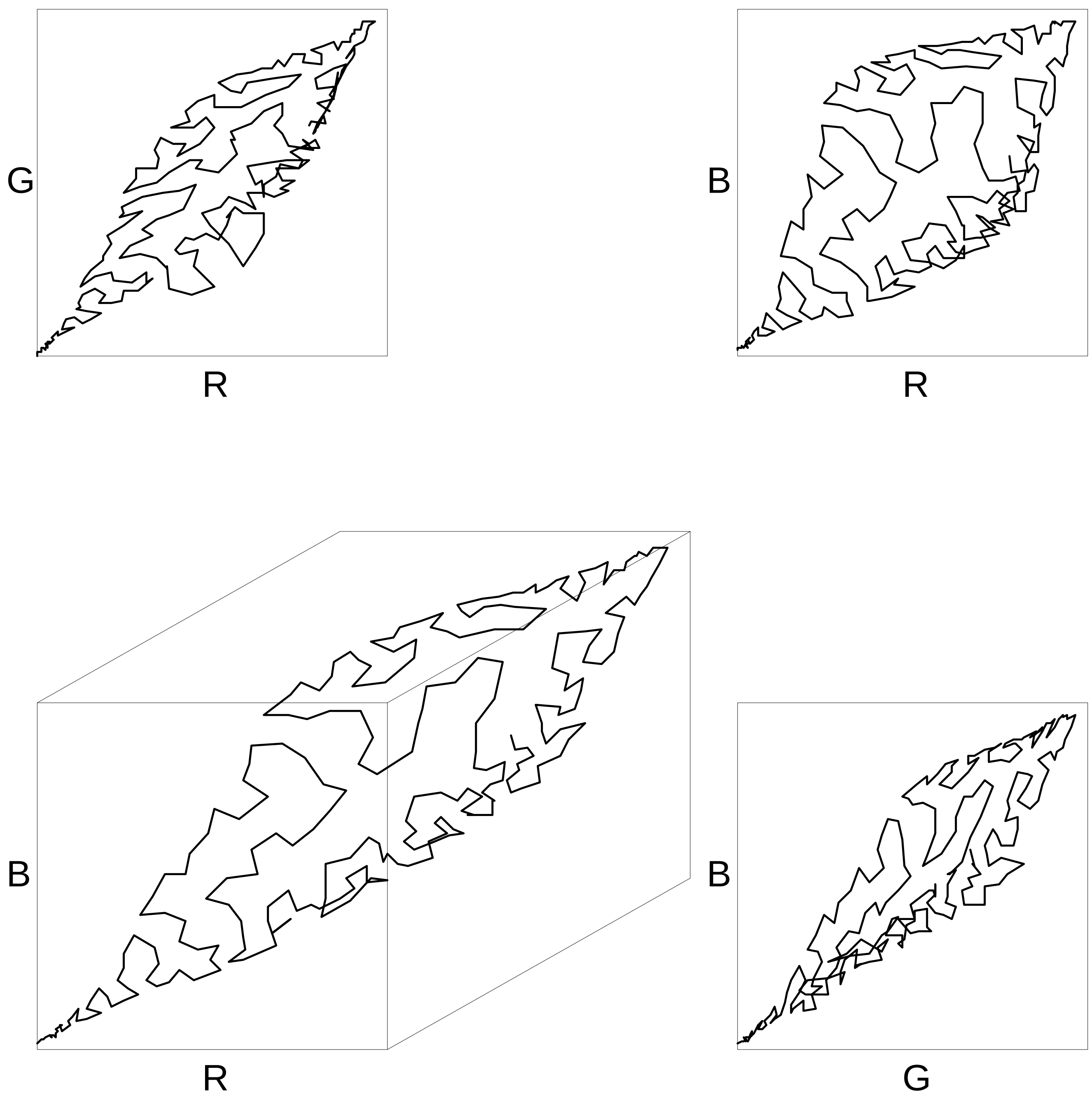


Figure 3: Kohonen Neural Network for an image with 122,227 distinct colours

a good colour quantization algorithm should deviate from choosing representatives with equal frequency in the case of isolated groups of pixels, and the network does precisely this, as we discuss below.

Initially the weight vectors are set with $R_i = G_i = B_i$. Kohonen [11, 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 as finer and finer adjustments are made. Kohonen [11, p 133] suggests decreasing α linearly, but we have found that for this and other applications 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:

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

For efficiency, we use an iterative approximation to this:

$$\alpha_0 = 1$$

$$\alpha_{t+1} = \frac{\lfloor \frac{29}{30}(2^{16}\alpha_t) \rfloor}{2^{16}}$$

Although this definition of α does not satisfy the guaranteed convergence criterion in [14, p 260], it performs extremely well for this application, requiring less training than is usual in Kohonen Neural Networks.

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 if $|i - j| = r$. We have found by experience that 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:

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

For efficiency, we again use an iterative approximation to this:

$$r_0 = 32$$

$$r_{t+1} = \frac{\lfloor \frac{29}{30}(2^6 r_t) \rfloor}{2^6}$$

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 $\rho_{(i,j,r)} = 0.75$).

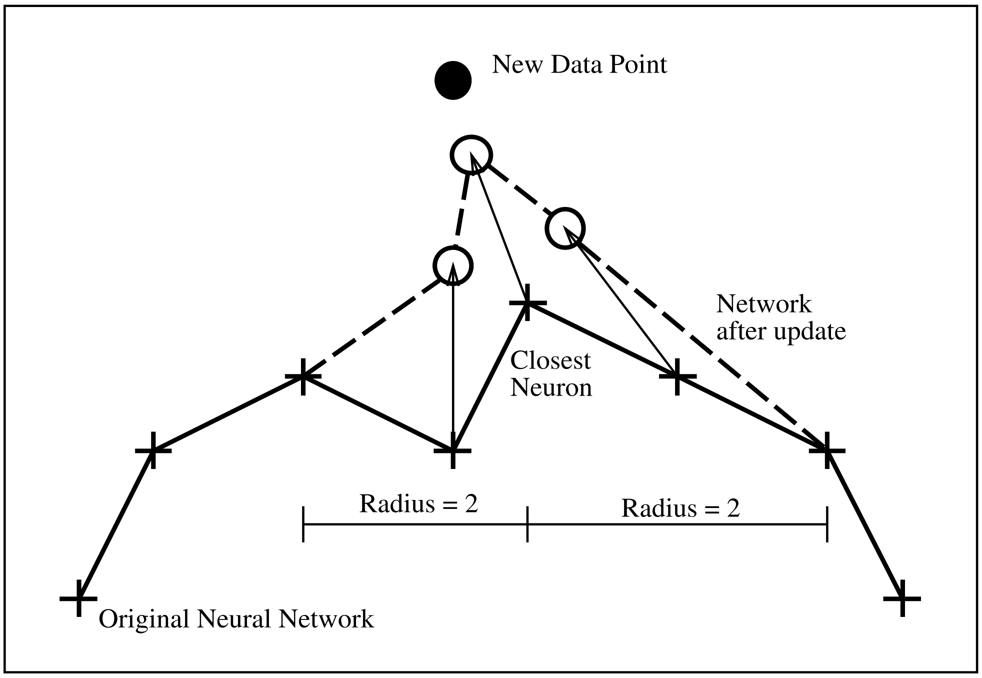


Figure 4: Kohonen Neural Network update algorithm

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$^{\text{st}}$ 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. This allows the algorithm to be used for a somewhat faster, but approximate, colour quantization process, by training the network on only a subset of the pixels in the image.

We have not yest defined the 'best' vector corresponding to an input. Kohonen [11, 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 [10, 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, 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 [10, 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 maximises speed, the weight vectors and the values of f_i and b_i are represented as rational approximations, performing all operations using integer arithmetic (and using shifting instead of multiplication where possible). The weight vectors are truncated to integers after training has been completed.

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.

The 'elastic' nature of the network ensures that the network forms a fractal space-filling curve [7], as shown in Figure 3. Together with the biasing scheme, it ensures that the neurons are relatively close together, i.e. that adjacent weight vectors represent similar colours, and that the network has the continuity properties discussed above.

3 Testing the Colour Maps

It is possible to combine the colour map produced by the Kohonen Neural Network with a Floyd-Steinberg dithering step [5, 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 to Kohonen Neural Network quantization algorithm and compared it with three existing quantization algorithms provided as part of the *xv* image processing package [4]. 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 [13], uses Sophisticated Median Cut without dithering. 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. This had the effect of halving the height and width of the images, but removing compression errors in the images. 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 three algorithms. In 9 out of the 12 images the smoothed error was also smaller, in spite of the fact that no dithering was used.

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
Kohonen Network	5.34	2.95

Table 2: Average Mean Mapping Errors for 12 Sample Images

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 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.

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.

Algorithm	CPU time (μs/pix)	
xvquick (dithered)	17	($\pm$ 18%)
xvslow (dithered)	72	($\pm$ 132%)
xvbest (non-dithered)	420	($\pm$ 201%)
Kohonen Network	1170	($\pm$ 4%)

Table 3: Average CPU time taken for 12 Sample Images

It can be seen that our algorithm is on average 3 times slower than *xvbest* and 16 times slower than *xvslow*. This is the penalty paid for the improved output image. There was less variation in time with our algorithm than with the other algorithms, so that in fact our algorithm ranged from 7 times slower than *xvbest* to slightly faster. The reason for the longer time taken is the use of exhaustive search through the network in order to find the best match to an input pixel.

The space overhead for our algorithm is very small: only 5 kB are needed to store the weight vectors and the b_i and f_i 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* space usage from the total *xvbest* and *xvslow* space usage. The results are shown in Table 4. The space used by *xvbest* depended on image size, with the line of best fit being approximately 400 kB of fixed overhead, plus 1 kB for each 400 pixels.

Algorithm	Space usage (kB)
xvslow (dithered)	521
xvbest (non-dithered)	1100–2100
Kohonen Network	5

Table 4: Average space usage for 12 Sample Images

In summary, our quantization algorithm using Kohonen Neural Networks gives significantly better output images than Sophisticated Median Cut, using significantly more time, but very little space. The low space overhead suggests that the algorithm would run significantly faster on a machine with a *copy-back* cache [16], where the network could be stored entirely in-cache. This would allow the weight vector updating process, as well as the search through the network for the best matching neuron, to be performed without memory references.

Although our quantization process is slow, there are two possible ways of more quickly producing an approximate output image for preview purposes. We have already indicated that training could be performed using only a subset of the pixels in the input image. It would also be possible to perform training with, say, only 51 neurons, thus using only $\frac{1}{5}$ of the normal training time. A set of 256 representatives could then be obtained by interpolating between the 51 weight vectors. This would be possible because the continuity properties of the network ensure that each neuron forms the centre of a cluster of pixels, and that the path joining adjacent neurons is as short as possible. Hence the line segment between two neurons usually passes through a region of the space occupied by pixels, and interpolation along that line segment provides possible additional cluster representatives. However, this would provide a less accurate result, since the pixels located in the 'gaps' between neurons would not be treated fairly.

4 Exploiting 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

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

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 (which is about 1% of the maximum possible mapping error). The mean mapping 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 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 1. 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. In fact, comparison with Figure 3 allows colour information to 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 in the following format:

(i) header (4 bytes)

(ii) colour map (768 bytes)

(iii) compressed false-grey image

We have compressed our test images in this way, using the JPEG compression algorithm [17, 8]. A 95% quality factor was used. For comparison, the original 24-bit image was compressed using the colour version of the JPEG algorithm. Table 5 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.

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

	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

Table 5: Comparison of Colour and False-Grey Compression

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 by 'dragging' the neural network through the colour space, in a similar way to the weight vector updating process shown in Figure 4. In order to do this, it would be convenient to represent the colour space in the HSV (Hue, Saturation, Value) format [5, p 333], rather than as an RGB cube.

5 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. Our algorithm produces a visibly superior result to the sophisticated version of Heckbert's Median-Cut algorithm, with almost half the mean mapping error, and with no contouring or other artifacts. The algorithm uses very little space (only 5 kB in addition to the space used to store the input image), but is on average 3 times slower than Median-Cut. Because of the low space usage, the algorithm would be expected to perform better in the presence of a *copy-back* cache.

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.

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] John Bradley. *xv - interactive image display for the X Window System*, Feb 1992. On-line UNIX Manual. Author's e-mail address: `bradley@cis.upenn.edu`

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

[6] 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.

[7] James Gleick. *Chaos: Making a New Science.* Cardinal/Sphere Books, 1987.

[8] Independent JPEG Group. *cjpeg - compress an image file to a JPEG file*, Feb 1992. On-line UNIX Manual. Author's e-mail address: `jpeg-info@uunet.uu.net`

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

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

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

[12] 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.

[13] Jef Poskanzer. *ppmquant - quantize the colors in a portable pixmap down to a specified number*, Jan 1991. On-line UNIX Manual. Author's e-mail address: `jef@well.sf.ca.us`

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

[15] 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.

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

[17] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, April 1991.