CIMCA 2003 Proceedings/ISBN 1740880684: M. Mohammadian (Ed.)
12-14 February 2003, Vienna - Austria

A New Approach for Compressing Color Images using Neural Network

A. K. M. Ashikur Rahman and Chowdhury Mofizur Rahman
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka-1000, Bangladesh
E-mail: ashikur@cs.ualberta.ca, cmr@cse.buet.edu

Abstract

In this paper a neural network based image compression method is presented. Neural networks offer the potential for providing a novel solution to the problem of data compression by its ability to generate an internal data representation. Our network, which is an application of counter propagation network, accepts a large amount of image data, compresses it for storage or transmission, and subsequently restores it when desired. A new approach for reducing training time by reconstructing representative vectors has also been proposed. Performance of the network has been evaluated using some standard real world images. It is shown that the development architecture and training algorithm provide high compression ratio and low distortion while maintaining the ability to generalize and is very robust as well.

1 Introduction

The technology trends are opening up both a capability and a need for digital continuous-tone color images. However the massive storage requirement for large numbers of high quality images is a technical impediment to widespread use of images. They are unsuitable to be transferred over slow communication line. The purpose of the image compression is to represent images with less data in order to save storage costs or transmission time and costs. However, the most effective compression is achieved by approximating the original image (rather than reducing it exactly).

There has been a considerable amount of activity in the area of image compression during the last thirty years [3, 4, 6]. An earlier approach to image compression is the run-length compression technique which is very familiar to all. Some neural network based image compression technique need to be mentioned here. Cottrel employed a two-layer neural network using the standard back propagation training algorithm to obtain image compression [1, 2]. However his method exploited only the correlation between pixel patches within each of the training patches, therefore only limited amount of compression could be achieved. Aran and Steven used multi-layer hierarchical network to compress image data [5]. They proposed a new training algorithm called nested training algorithm to make the training process faster. However they are also limited only to the back propagation technique. They have worked only with the gray scale images and the compression ratio achieved using their algorithm is 8:1.

In this paper we have proposed a new method by clustering the vectors onto limited number of clusters based on compression requirement. We have used counter propagation network, which we think would be the best for our application. Our compression technique is not loss less, however we have proposed a compression technique with less amount of distortion.

2 Methodology

In a truly uncompressed bitmap file (BMP file), the color information of an image is stored pixel by pixel and their position in images is immaterial because they are stored in a predetermined order. The color information of a pixel is basically defined by its three-color quantities: red, green and blue quantity. By varying the contents of these three quantities

various pixels are formed. In a bitmap file 24 bits are allocated for storing each pixel's color information. Each color quantity is basically of 8 bits thereby needs 24 (3×8) bits in total. Now 24 bits for a particular pixel means there are 2^{24} (= 16 million) possible colors. During compression we should allow lesser number of bits as much as possible to make compression process effective. The problem is, if we allow lesser number of bits per pixel during compression, then we may not be able to allow such huge number of colors like 16 million. In that case we have to reduce the total color requirement and thereby we must prepare ourselves to except some loss in the color information. As for example, if we allow 10 bits per pixel during compression then we are basically allowing 2^{10} (= 1024) colors only. Images can always tolerate some amount of noises if they can be organized in a very tactful manner.

So we need to reduce total number of colors of an image to achieve compression effectively. One way to achieve this objective is to group those pixels together, which are exactly same or very close to each other with respect to their color information. If we need to design an algorithm which will automatically cluster the similar pixels together then there should be some precise underlying definition of similarity between colors.

2.1 Mathematical definition of similar pixels

The total 24 bits required to store the color information of a pixel are divided into three equal 8 bits part. Each 8-bit quantity represents the red, green and blue quantity and each quantity may vary from 0 to 255. Now if we interpret three color quantities as three individual dimensions on a three-dimensional space then each pixel may be viewed as a vector lying on a three-dimensional space. Figure 1 shows this diagrammatically.

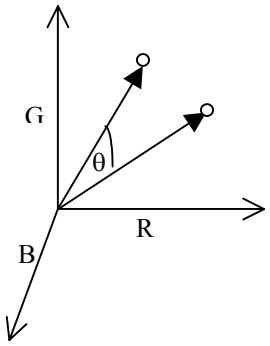


Figure 1: Pixels on 3-D space.

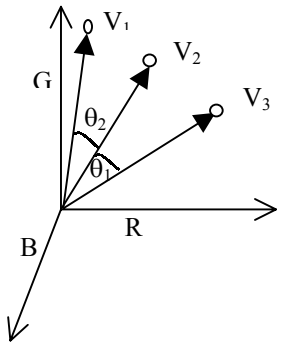


Figure 2: Three pixels are mapped onto three vectors

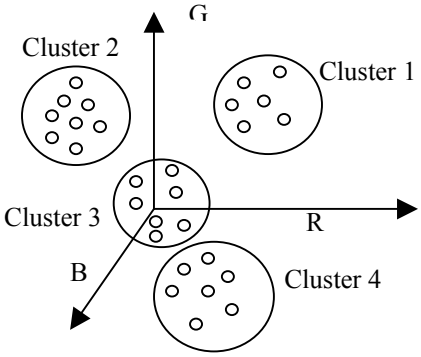


Figure 3: Clustering of pixels based on their vectors

Now let us consider about two pixels defined by their color content. If we map them on a three-dimensional space as shown in Figure 1 then they will constitute two separate vectors having an angle θ in between them. We can easily use this angle θ for defining similarities between colors. As for example let us consider three pixels p_1, p_2, p_3 which are mapped onto three individual vectors $\mathbf{V_1}$, $\mathbf{V_2}$, $\mathbf{V_3}$ respectively as shown in Figure 2. Now, let $\theta_1 > \theta_2$. Then obviously the vector $\mathbf{V_2}$ is much closer to the vector $\mathbf{V_3}$ then to the vector $\mathbf{V_1}$. So pixel p_2 will be much similar to the pixel p_3 then to the pixel p_1 with respect to their color content. In this way we can easily map each and every pixel of an image onto this three-dimensional space and try to find the similarities between them. The pixels will be grouped together based on their similarities and number of groups will depend on the required compression level. This phenomenon has been shown pictorially in Figure 3. Here all pixels of an image have been mapped on a three-dimensional space. One thing is clearly evident

from the picture that although there are many pixels on an image, but all pixels of an image may be grouped together into small number of groups. The ultimate quality of the image will basically depend upon the total number of allowable groups. As for example if we allow only a single group then all pixels are bound to fall upon this single group thereby degrading the quality of the image. But if we allow more number of groups then obviously the overall picture quality will be improved. In that case the space requirement will be increased. So there is a trade- off between total number of groups and total space requirement and we must find a balance in between the two. This issue is one of the major concerns of this paper and will be explored experimentally.

2.2 Rough sketch of the overall procedure

The whole compression-decompression procedure may be executed in four major steps:

1. Clustering all the pixels into predetermined number of groups.
2. Producing a representative color for each group.
3. For each pixel storing only the cluster number during compression.
4. During decompression restoring the cluster number and storing the representative color of that cluster.

In the subsequent subsections each of the five major steps are described in details.

2.3 Clustering process

For clustering purpose we have taken the help of neural network and we have found that counter propagation network is best suited for our purpose. Counter propagation networks are very popular because of its simplicity in calculations and strong power of generalization. However other networks such as backpropagation network may also be used for clustering purpose, but the major disadvantage of backpropagation network is its training time is higher because of the complexity in its equations.

For our purpose we have used the counter propagation network shown in Figure 4. The total number of neurons in the input layer will be exactly three as each input is basically a vector having three constituent parts (e.g., the red, green and blue quantity). For particular pixel input the neuron I_1 will accept the amount of red quantity, the neuron I_2 will accept the

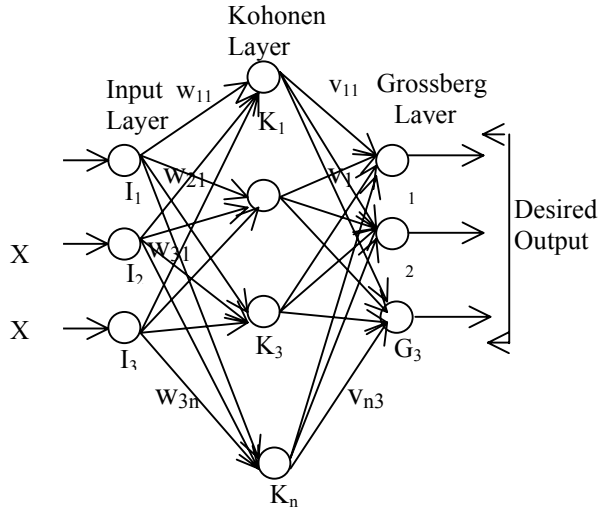


Figure 4: The counter propagation network used for clustering

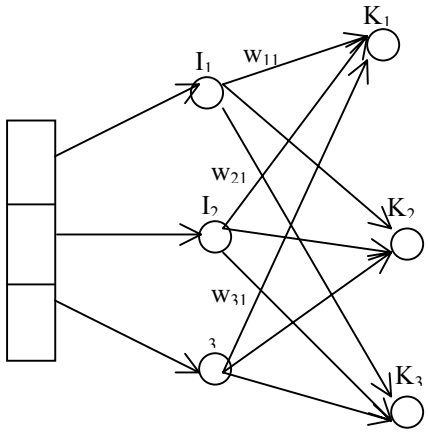


Figure 5: The first two layers of the counter propagation network.

amount of green quantity, the neuron I_3 will accept the amount of blue quantity.

The number of neurons in the Kohonen layer will vary based on the total number of clusters we allow. If we allow 1024 clusters then there are exactly 1024 neurons and if we allow 256 clusters then there are 256 neurons. Each of the neuron in the middle layer has connecting weights for each of the neuron of the previous layer. As for example the first neuron K_1 of the middle layer has three connecting weights w_{11}, w_{21}, and w_{31} with the neuron I_1, I_2, and I_3 of the previous layer respectively. The connecting weights are doing all the tricks for clustering the pixels into groups. They will be trained using real life images in unsupervised mode. After training all the weights will be finally adjusted to a point on which they can distinguish the similar pixels from the dissimilar pixels.

The neurons in the final layer are responsible for reproduction of the color information of pixels. As there are three constituent parts of a pixel that is why there are exactly three neurons in this layer. Each of the three neurons is connected by some weights to each of the neurons of the previous layer. The connecting weights will be trained up in such a way so that they can produce an average of all the colors within a group.

The first two layers are shown separately in Figure 5. Let us consider that a new image has been given as input for compression. We have to retrieve the pixel information one after another and we will apply the color information of each pixel to our network. In the following paragraph the clustering process for a particular pixel is described. This process will be repeated for each pixel of the image.

A particular pixel X has three constituent parts (e.g. the RGB values). Let the three constituent parts be x_1, x_2 and x_3. x_1 is the amount of red quantity and will be applied to neuron I_1 of the first layer. x_2 (the amount of green quantity) and x_3 (the amount of blue quantity) will be applied to neurons I_2 and I_3 respectively. After applying the input we have to calculate the output of each neuron in the middle layer. For a particular neuron say K_1 the output calculation is as follows. The neuron K_1 has three weights associated with it, firstly the weight w_{11} between I_1 and K_1, secondly the eight w_{21} between I_2 and K_1, and finally the weight w_{31} between I_3 and K_1. The output of neuron K_1 is named as Net_1 and will be calculated by using the following equation:

$$Net_1 = w_{11}x_1 + w_{21}x_2 + w_{31}x_3 \quad \text{.......................................}(1)$$

In this way each neuron of the middle layer has its own connecting weights and for each neuron j of the middle layer we have to calculate its output Net_j:

$$Net_j = w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3 \quad \text{.......................................}(2)$$

After calculating all the outputs of the neurons of the middle layer the neuron having maximum output will be declared as the “winning neuron” for the current pixel. The current pixel will be assigned to that neuron *j* for which the output is maximum. When a pixel having the similar type of color information will be applied to the input layer, the same neuron is going to fire (e.g., the same neuron’s output will be maximum and for a pixel with different type a different neuron is going to be fired). In this way the clustering process will be fully automated. The Algorithm 2.1 describes the overall procedure of clustering in brief.

Algorithm 2.1:

1. **for** each pixel **X** = (x_1, x_2, x_3) **do**:
2. Apply the constituent part of the color info. to respective neurons of input layer.
3. **for** j = 1 to n **do**:
4. Multiply input vector component with weight vector component and calculate the output NET_j using the following equation:

$$NET_j = \sum_i x_i w_{ij}$$

5. Assign the pixel on that group *j* for which the value of NET_j is maximum.

2.4 Producing the representative color of each group

The final layer (also known as Grossberg layer) of the counter propagation network is responsible for reproduction of the colors of the pixels. Truly speaking, the representative color for a particular cluster will be the average of all the colors within the cluster. The generation of representative color will take place at the time of clustering process. For a particular pixel X at first we have to find the winning neuron on which the pixel falls. Only the connecting weights between the winning neuron and the three neurons of the final layer will be adjusted. The equation for weight adjustment is as follows:

$$V_{ij}(new) = V_{ij}(old) + \beta(y_j - V_{ij}(old)) \quad \ldots\ldots(3)$$

Where,

V_{ij}(new) = new weight between neuron *i* of the middle layer and neuron *j* of the final layer.
V_{ij}(old) = old weight between neuron *i* of the middle layer and neuron *j* of the final layer.
β = a scalar quantity.
y_j = the j-th component of the input vector (remember that there are three constituent parts of an input pixel).

In Equation 3 actually the amount of the weight adjustment is proportional to the difference between the weight and the desired output of the Grossberg neuron to which it connects. In summery, the similar pixels will fire the same Kohonen neuron and the weights connected to that firing neuron will be adjusted so that it gives an average color of those similar color pixels. The network parameter β is actually a scalar quantity and its value is not constant. We have to apply all the pixels of a particular image for several times until the weights between the Kohonen and Grossberg layer converge. During this time the value of β will be increased slowly. Initially its value should be very small say 0.1. Gradually this value should be increased in small steps until a sufficient large value say 0.25. Thus it will help the convergence process of the weights between the last two layers. One thing should be mentioned here, although we are training the weights between the last two layers, during normal operation of compression one might think that the process time could be longer. Actually the weight convergence time of these weights is very small (about 20-30 seconds) and one need not to be worried too much.

2.5 Compression process

The main problem of the clustering algorithm is, it is basically clustering on the basis of angular distance of the color vectors ignoring the actual length of those color vectors. Consider two pixels p_1 (255, 255, 255) and p_2 (4, 4, 4). After normalisation they will form two vectors $\mathbf{V_1}$ (1/√3, 1/√3, 1/√3) and $\mathbf{V_2}$ (1/√3, 1/√3, 1/√3). So obviously they are going to fall on the same clusters although actually these two pixels are having very dissimilar color information. The second problem is, all constituent parts of a color vector are positive. So every of them will fall on the 1st quadrant of a three-dimensional space leaving other quadrants totally unused. As a result the distribution of the color vectors will be very dense over the first quadrant whereas they might be distributed uniformly over the whole three-dimensional space.

To remove these problems, we have taken two necessary steps. At first we have deducted 127 from each color value of each pixel. This will cause some constituent parts of the color vector to become negative, thereby forcing some color vectors to fall on the other quadrants of the three-dimensional space. As for example the previously considered two pixels were p_1 (255, 255, 255) and p_2 (4, 4, 4). Now if we deduct 127 from each part of the

color information, the pixels will be transformed into p_1(128, 128, 128) and p_2(-123, -123, -123). After normalisation they will form two vectors $\mathbf{V_1}$ (1/√3, 1/√3, 1/√3) and $\mathbf{V_2}$ (-1/√3, -1/√3, -1/√3). Now they will fall on two different quadrant namely first and eighth quadrant. So this time they will fall on two separate clusters and the average error will be drastically reduced.

To implement the above-mentioned procedure, a new problem now arises. The weight vectors were trained up only for the first quadrant and not for the other quadrants. So this needs a retraining of the weight vectors. But if we apply the following procedure then the retraining procedure is no longer needed.

To avoid the need for retraining, we may easily exploit the symmetrical property of a sphere. At first let us consider a two-dimensional space. Suppose the weight vectors are trained in the first quadrant and after training, these weight vectors are uniformly distributed over the first quadrant.

Now we need some more weight vectors for the other quadrant. The circle is basically symmetrical over its two axes. So we don't need a further training. Instead, we can just construct the four vectors in the second quadrant from the four vectors of the first quadrant just by taking the negative of the x values of the each weight vectors of the first quadrant. We can construct four other weight vectors in the third quadrant if we just take negative of the x and y values of the weight vectors in the first quadrant. Again four other vectors may be constructed in the fourth quadrant just taking the negative of the y values. Figure 6a- 6d shows all these construction process.

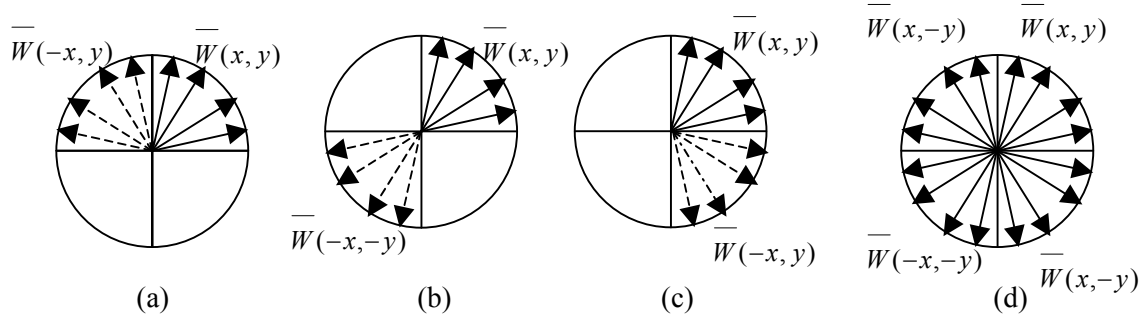


Figure 6: (a) Constructing weight vectors for the second quadrant by taking negative of x values
(b) Constructing weight vectors for the third quadrant by taking negative of x and y values.
(c) Constructing weight vectors for the fourth quadrant by taking negative of y values.
(d) Final construction of 16 weight vectors.

The basic idea is shown above in a two dimensional space. This idea may be easily extended for the three dimensional space., because in a three dimensional space there will be a sphere instead of circle and a sphere is also symmetrical about its three separate axes.

Before compression we need to extend our weight vectors. In a three-dimensional space there will be eight quadrants. We have the weight vectors for only the first quadrant. In the first quadrant all the values of x, y and z are positive. Say these weight vectors are of the form $\mathbf{W_{000}}$ (x, y, z). Now let us extend these weight vectors for the second quadrant. In the second quadrant, the value of x is negative and the values of y and z are positive. So in the second quadrant the weight vectors will be constructed by using the following equation:

$$\mathbf{W_{001}}\ (x, y, z) = \mathbf{W_{000}}\ (-x, y, z) \quad \ldots\ldots(4)$$

Similarly for other six quadrants the weight vectors will be:

$$\mathbf{W_{010}}\ (x, y, z) = \mathbf{W_{000}}\ (x, -y, z) \quad \ldots\ldots(5)$$

$$\mathbf{W_{100}}\ (x, y, z) = \mathbf{W_{000}}\ (x, y, -z) \quad \ldots\ldots(6)$$

$$\mathbf{W_{011}}\ (x, y, z) = \mathbf{W_{000}}\ (-x, -y, z) \quad \ldots\ldots(7)$$

$$\mathbf{W_{101}}\ (x, y, z) = \mathbf{W_{000}}\ (-x, y, -z) \quad \ldots\ldots(8)$$

$\mathbf{W_{110}}$ (x, y, z) = $\mathbf{W_{000}}$ (x, -y, -z) ……………………………………(9)
$\mathbf{W_{111}}$ (x, y, z) = $\mathbf{W_{000}}$ (-x, -y, -z)… ………………………………(10)

Algorithm 2.2 describes the new compression algorithm.

Algorithm 2.2:

1. Extend the weight vectors using equations 4 to 10.
2. **for** each pixel **X** = (x_1, x_2, x_3) **do**:
3. Transform each pixel by deducting 127 from each part and then apply (i.e., $\mathbf{X}^{/}$ = (x_1-127, x_2-127, x_3-127)
4. Let q be the quadrant number on which current pixel falls after transformation.
5. Let there be n number of neurons in each quadrant.
6. **for** j = 1 to n **do**:
7. Multiply input vector component with weight vector component and calculate the output NET_{qj} using the following equation:

$$NET_{qj} = \sum_i x_i w_{qij}$$

8. Assign the pixel on that group j of quadrant q for which the value of NET_{qj} is maximum.
9. Update the weights V_{qp1}, V_{qp2} and V_{qp3} using the following equation:

$$V_{qpj}(new) = V_{qpj}(old) + \beta \times \left(x_j - V_{qpj}(old) \right)$$

10. Step 2 will be repeated until weights V_{pij}'s are converged.
11. Generate the output file (the compressed file) as follows:
    i) At first store the header portion of the input file.
    ii) Next store the weights between the middle layer and the final layer.
    iii) Store the quadrant number after using run length encoding. Finally for each pixel store the index of the winning neuron.

2.6 Decompression process

The decompression algorithm will accept the compressed file and will produce the uncompressed bitmap file. The compressed file contains the header portion of the original file and this header portion will be restored at first during decompression. From this header part we can retrieve the height and width information of the original image. So total number of pixels will be calculated by multiplying height and width. Next the weights between the middle layer and the final layer will be loaded from the compressed file in sequential order.

Algorithm 2.3:

1. Retrieve the header portion of the compressed file.
2. Load the weights between the middle layer and the final layer from the compressed file.
3. Let n be the total number of pixels determined by multiplying height by width.
4. For n pixels retrieve the quadrant number by applying the reverse algorithm of run length encoding.
5. For n pixels retrieve the index of the winning neuron.
6. **for** i = 1 to n **do**:
   Determine the actual winning neuron by using the combination of quadrant number and cluster number payer. Let the quadrant number be q and the cluster number be p. Find the weights $V_{qp1,}$ V_{qp2} and V_{qp3} and concatenate these three numbers to constitute color information of the corresponding pixel.
7. Generate the original bitmap file as follows:
    i) At first store the header portion.
    ii) For each pixel store the color information as described in step 6.

The weights between the input layer and the middle layer will be loaded from the weight files saved during the training process.

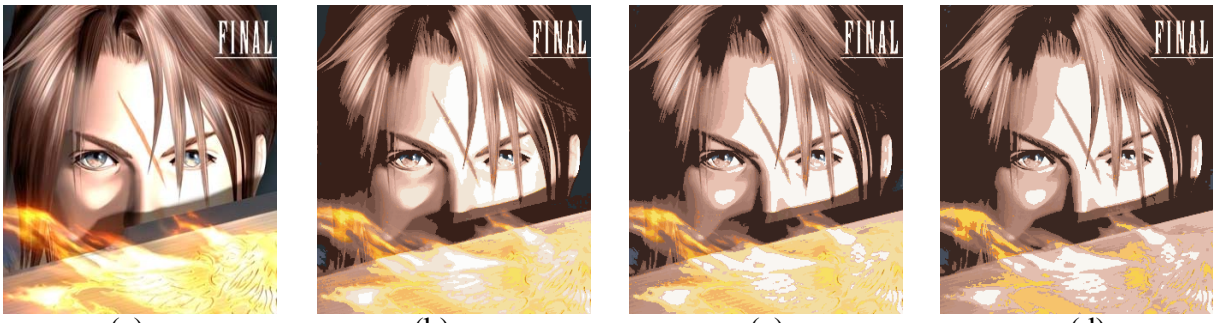
During compression the quadrant number of each pixel will be stored using run length encoding. Now during decompression applying a reverse algorithm of run length encoding will retrieve this quadrant number. Next during compression the cluster number of the winning neuron was stored pixel by pixel. Now during decompression this cluster number will be retrieved for each pixel. For each pixel, using the combination of quadrant number and the cluster number the actual winning neuron will be determined and the three connecting weights between the wining neuron and the neurons of the final layer will comprise three parts of the color information (red, green and blue) of each pixel. Algorithm 2.3 describes the complete decompression algorithm.

3 Experimental results

The whole network architecture was implemented and tested with various training and test images. In this section we are going to describe some experimental results obtained from our network. The image shown in Figure 7(a) was used for training our network. After compression and decompression procedure the final image obtained from our network is shown in Figure 7(b). Similarly our network was trained up using the same image by restricting the total number of clusters. Figure 9 and 10 shows the picture obtained using 128 (network was trained up for 16 clusters) and 64 colors (network was trained up for 8 clusters) respectively. For all these three cases the quality of the picture obtained after compression-decompression procedure is quite acceptable. To investigate the generalisation capability of our network, we have selected some test images and applied them to our network. We have applied standard "Lena" image and the "Mandrill" image shown in Figure 8(a) and 9(a) respectively. As you can see for both of the picture our algorithm worked well. Clearly this indicates about the generalisation capability of our network. The compression ratios achieved with various color restrictions are shown in Table 1. If we closely observe on images obtained with various compression ratios for all the images, then certainly we find the quality fall with increasing compression ratios. Actual quality measure is PSNR (Peak-Signal-to-Noise-Ratio) and is shown in Table 2 with various compression ratios for all the images.

Images	Total colors allowed	Original file size	Space required to store quadrant number	Space required to store cluster number	Compressed file size	Compression ratio
Squall	256	300KB	20.6KB	64KB	85KB	3.8:1
	128	300KB	20.6KB	51KB	72KB	4.2:1
	64	300KB	20.6KB	38KB	59KB	5:1
Lena	256	70KB	6.29KB	14.44K	20.73KB	3.37:1
	128	70KB	6.29KB	11.56KB	17.85KB	3.92:1
	64	70KB	6.29KB	8.67KB	14.96KB	4.68:1
	32	70KB	6.29KB	5.77KB	12.06KB	5.80:1
	16	70KB	6.29KB	2.89KB	9.18KB	7.62:1
Mandrill	256	44.1KB	7.88KB	9.4K	17.28KB	2.55:1
	128	44.1KB	7.88KB	7.33KB	15.21KB	2.90:1
	64	44.1KB	7.88KB	5.6KB	13.48KB	3.27:1
	32	44.1KB	7.88KB	3.75KB	10.63KB	4.14:1
	16	44.1KB	7.88KB	1.8KB	8.68KB	5.08:1

Table 1: Showing size of various compressed file



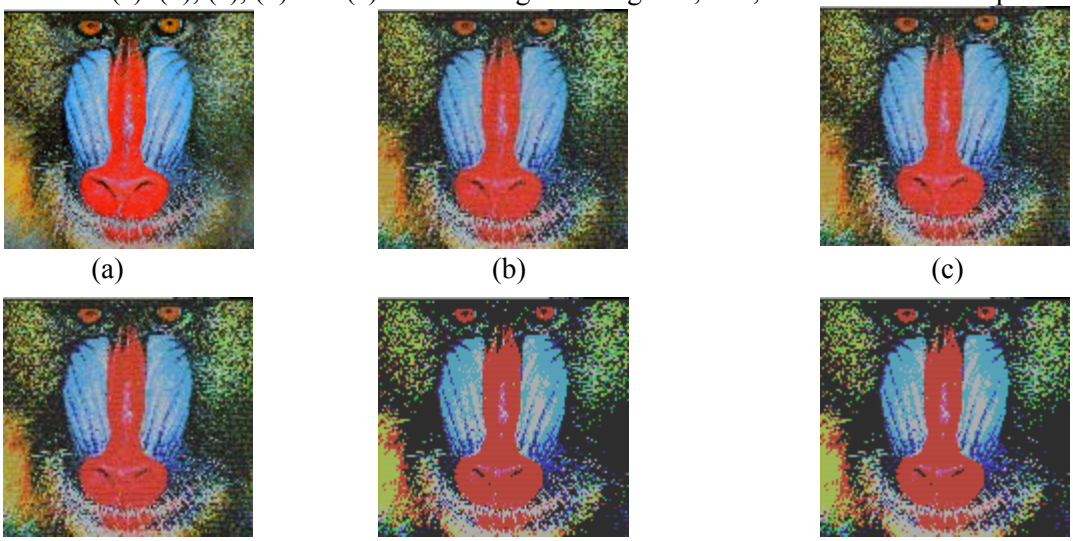
(a) (b) (c) (d)

Figure 7: The Original file "Squall" is shown in (a). (b), (c), (d) are image obtained using 256, 128 and 64 colors respectively.

(a) (b) (c)

(d) (e)

Figure 8: Showing the quality of the image after compressing the standard "Lena" image in (a). (b), (c), (d) and (e) are the images having 256, 128,64 and 32 colors respectively.



(a) (b) (c)

(d) (e) (f)

Figure 9: Showing the quality of the image after compressing the standard "Mandrill" image in (a). (b), (c), (d), (e) and (f) are the images having 256, 128,64, 32 and 16 colors respectively.

The peak-signal-to-noise-ratio (PSNR) is calculated in three-color quantities separately using the Equations 11 and 12:

$$PSNR_{(color)} = 10\log 10\left(\frac{255^2}{MSE_{(color)}}\right)dB \quad \text{.....(11)}$$

$$\text{and } MSE_{(color)} = \frac{1}{N}\sum_{j=1}^{N}\left(x_{jk} - y_{jk}\right)^2 \quad \text{.....(12)}$$

for each *color* = red, green and blue. In the equations, x_{jk} is the corresponding color component's quantity of the original pixel and y_{jk} is the corresponding color component's quantity of the pixel after compression and decompression procedure. The final value of PSNR is calculated by taking average of three PSNR.

$$PSNR = \frac{PSNR_{(red)} + PSNR_{(green)} + PSNR_{(blue)}}{3} \quad \text{.....(13)}$$

Images	Compression ratio	$PSNR_{(red)}$ in dB	$PSNR_{(green)}$ in dB	$PSNR_{(blue)}$ in dB	PSNR in dB
Squall	3.8:1	23.12727	22.75904	22.49132	22.79255
	4.2:1	22.36818	21.75789	21.89486	22.00697
	5:1	21.69062	20.01077	19.23142	20.31094
	6.6:1	20.70212	18.04946	17.54719	18.76626
Lena	4.68:1	22.04484	22.01807	23.91896	22.66062
	5.80:1	21.70118	21.28969	22.9719	21.98759
Mandrill	2.55:1	21.07949	21.06464	21.86561	21.33658
	2.90:1	20.83268	20.1024	21.72429	20.88646
	3.27:1	19.89669	19.56891	20.8883	20.11797
	4.14:1	19.41057	19.19419	20.60719	19.73732
	5.08:1	17.77385	17.85663	17.91771	17.8494

Table 2: Showing PSNR of some images with various compression ratios using our algorithm.

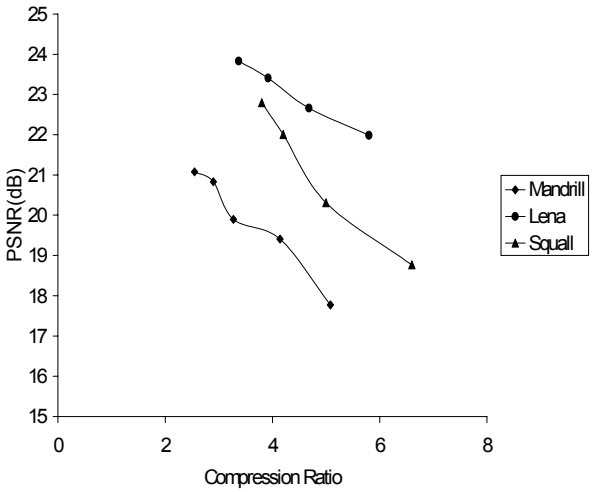


Figure 10: Showing PSNR of various images with various compression ratios.

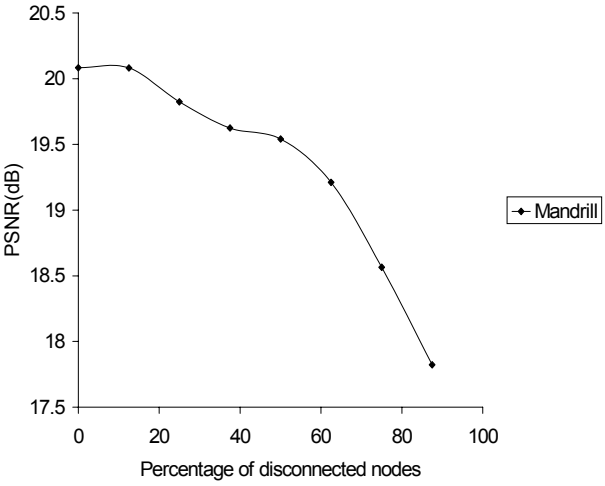


Figure 11: Showing robustness for "Mandrill" image

Figure 10 shows the PSNR curve for all three images. All curves are decreasing in PSNR with the increase in compression ratio. As for example the curve for "Mandrill" image

varies from 21.34 dB with compression ratio of 2.55:1 to 17.85 dB with compression ratio of 5.08:1. So a change of 2.47 in compression ratio causes a quality fall of 3.49 dB in PSNR, which is quite acceptable. The image "Lena" shows highest value in PSNR for every compression ratios with respect to all other images. The PSNR curves for the "Lena" image varies slowly than other two PSNR curves for the "Mandrill" and the "Squall" image. It clearly indicates that the "Lena" image may be compressed further down with very insignificant fall in PSNR, whereas further compression in the "Squall" image and the "Mandrill" image may cause a significant fall in quality.

4 Robustness

To illustrate the robustness of our algorithm we have experimented with the "Mandrill" image. At first we have chosen the network with 8 clusters per quadrant (thereby allowing 64 colors) and then we disconnected the neuron number 0 and all its connecting weights with input layer and the final layer for each quadrant. Similarly we have disconnected the neuron number 1, 2, 3, 4, 5, 6 and 7 of each quadrant. Table 3 shows how much quality falls occurred by doing such disconnection. Clearly the loss in PSNR is very small (less then 0.4 dB). In fact by disconnecting single neuron in each cluster actually in total 8 neurons in 8 quadrant was off and in fact 12.5% clusters was off, but the change in PSNR is less then 0.4 dB. So our network is very robust also. After that we have experimented with same "Mandrill" image, this time with 32 colors and we disconnected single neuron of 4 neurons from each quadrant. Table 4 shows the loss in PSNR. This time, 25% neurons were disconnected and the PSNR fall is less then 0.7 dB. Clearly our network is robust.

Figure 11 shows the effect on PSNR with various percentages of disconnected neurons. Clearly the curve is decreasing with increasing number of disconnected neurons. The change in PSNR varies in small steps. By disconnecting from 0% to 80 % of total neurons in the middle layer, the PSNR fall is between 0 to 3.5 dB. Table 5 lists the value of PSNR in dB for various percentages of disconnected nodes.

Images	**Disconnected Neuron Number**	**$PSNR_{(red)}$ in dB**	**$PSNR_{(green)}$ in dB**	**$PSNR_{(blue)}$ in dB**	**PSNR in dB**
Mandrill	None	19.86457	19.54126	20.84359	20.08314
	0	19.86043	19.51591	20.76448	20.04694
	1	19.67199	19.17281	20.73036	19.85839
	2	19.5899	19.2887	20.74079	19.87313
	3	19.85581	19.52733	20.84014	20.07443
	4	19.48934	19.78766	19.9352	19.7374
	5	19.83374	19.51884	20.83287	20.06182
	6	19.8228	19.50572	20.82248	20.05033
	7	19.70749	19.41461	20.34585	19.82265

Table 3: Showing PSNR by disconnecting 12.5% neurons for "Mandrill" image.

Images	**Disconnected Neuron Number**	**$PSNR_{(red)}$ in dB**	**$PSNR_{(green)}$ in dB**	**$PSNR_{(blue)}$ in dB**	**PSNR in dB**
Mandrill	None	19.3798	19.16469	20.56916	19.70455
	0	20.49416	19.3916	18.35762	19.41446
	1	19.18535	18.72458	20.46544	19.45846
	2	18.19824	18.65634	20.1941	19.01622
	3	19.18636	19.03782	19.38286	19.20235

Table 4: Showing PSNR by disconnecting 25% neurons for "Mandrill" image.

Images	% of Disconnected Neurons	$PSNR_{(red)}$ in dB	$PSNR_{(green)}$ in dB	$PSNR_{(blue)}$ in dB	PSNR in dB
Mandrill	0%	19.86457	19.54126	20.84359	20.08314
	12.5%	19.89252	19.54354	20.8087	20.08159
	25%	19.66803	19.14951	20.65326	19.8236
	37.5%	19.40246	18.89168	20.57694	19.62369
	50%	19.37933	18.87414	20.36664	19.54004
	62.50%	20.48286	18.66205	18.48897	19.21129
	75%	18.62616	18.35719	18.70977	18.56438
	87.5%	17.74475	17.83206	17.89094	17.82259

Table 5: Showing PSNR at various percentages of disconnected neurons for "Mandrill" image.

5 Conclusions

A counter propagation neural network based image compression technique has been used to successfully compress and decompress image data. A quality improvement technique together with faster training procedure has also been proposed. Additional simulation runs using other types of images (e.g. a building, outdoor scenes) were also performed yielding similar results to those presented here. Thus when all the network parameters and network sizes are properly specified, the network has the ability to both learn and generalise over a wide class of images. The network also shows robustness for various classes of images. Based on these considerations, this neural network architecture should be considered as a viable alternative to other more traditional techniques, which are currently used.

This work may be extended in several ways. The whole image may be subdivided into small part of sub-images and then our algorithm might be applied to evaluate performance. Also the correlation between sub-images may be easily exploited to achieve further compression. A hierarchical counter propagation network might also be employed to get some effective results.

References

[1] Cottrell, G.W., Munro, P., and Zipser, D.: 'Image compression by back propagation: An example of extensional programming', *Advances in Cognitive Science*, 1985, **1**, pp. 209-239,

[2] Cottrell, G.W., Munro, P., and Zipser, D.: 'Learning internal representation from gray scale images: An example of extensional programming', Proceedings of the Ninth Conference of the Cognitive Science Society, 1988, pp. 462-473

[3] Grossberg, S.: 'Embedding fields: underlying philosophy, mathematics and applications of psychology, physiology and anatomy', *Journal of Cybernetics*, 1971, pp. 28-50

[4] Jain, A.K.: 'Image data compression: A review', Proceedings of the IEEE, 1981, **69**, pp. 349-389

[5] Namphol, A., Steven, H. C., and Arozullah, M.: 'Image compression with a hierarchical neural network', *IEEE Transactions on Aerospace and Electronic Systems,* 1996, **32** (1), pp. 326-337

[6] Newaz, M.S. Rahim, and Takashi, Yahagi: 'Image compression by new sub-image block classification techniques using neural networks', *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2000, **E83-A** (10), pp. 2040-2043