



## **Digital Watermarking Of Cover Image Using Weights Generated From Training A Neural Network With Fingerprint Image**

Veena Desai <sup>1</sup> Sunayana K <sup>2</sup>, Dr. D H Rao <sup>3</sup>

<sup>1</sup>Research Centre, GIT, Belgaum, India

<sup>2</sup>KLECET, Belgaum, India

<sup>3</sup>Jain College of Engineering, Belgaum, India

**Abstract:** In this paper we propose a novel method of hiding an image in a cover image using weights generated by the neural network. A neural network is designed and trained with the cover image and fingerprint image as target. The weights generated by the network are embedded in the cover image itself using a different embedding method to form a watermarked image. The PSNR of the original image and watermarked image is calculated. The weight generation and embedding algorithm has been applied on three sample images a) Lena b) Apollo c) Nature and their PSNR calculated. The average PSNR of the three samples is found to be 45dB which provides good imperceptibility of the watermarked image. After applying the weights extraction algorithm the weights are compared and found to be the same as the generated weights.

**Keywords:** Digital watermarking, neural network, hidden weights, cover image, fingerprint image

### **I. Introduction**

Digital Watermarking has evolved as one of the important technologies for digital media protection. Many techniques based on spatial and frequency domain have been developed in the recent past and are being used for effective watermarking [1-2]. However, there is always a trade-off between robustness and imperceptibility features of watermarking offered by these techniques. Several papers offer different techniques for watermarking using neural networks[3-8] This paper offers a technique based on Back propagation Neural Network to train a given cover image to produce a desired watermark image. At the end of the training, the entire trained neural network weights have been successfully hidden within the cover image itself. This makes it possible to supply only the cover image without any external weight files. By extraction techniques, the weights can be derived from the cover image and used to reconstruct the trained Neural Network. The technique for hiding the weights into the cover image has been designed in such a way that it does not produce visual deterioration of the original cover image. This method is extra secure as it leads to watermarking indirectly.

We use a Back propagation Neural Network to hide the cover image. After the neural network is trained, the trained network weights have been inserted into the cover image itself by using a special technique which preserves the visual quality of the original cover image.

### **II. Implementation Details**

The neural network is created with one input layer, one hidden and one output layer of neurons. Since fingerprints are images used in biometrics and often require to be concealed a fingerprint image has been used as concealed image. The cover images chosen are Lena, Nature and Apollo. These images are used to generate the weight file for fingerprint after training the network. The weights are then embedded in the cover image itself using a new watermark technique. At the receiver side the weights are retrieved from the watermarked image.

As the hidden weight matrix is inside the watermarked image, and the Neural Network can be reconstructed only after successfully deriving the weight matrix from the watermarked image. Non authentic watermarked images fail to reconstruct the Neural Network and produce the watermark image. Thus, authenticity feature is preserved in this scheme.

#### **A. Embedding**

- 1) The target watermark image is taken to serve as output to a Back propagation Neural Network.
- 2) A Back propagation Neural Network is chosen with 1 input, 1 hidden and 1 output layer.
- 3) The cover image is supplied as input to the input layer of the network, and weights are adjusted to produce the corresponding target watermark image at the output layer using Back propagation algorithm.
- 4) The trained weight matrix is hidden within the cover image itself by using special technique described in algorithm below.

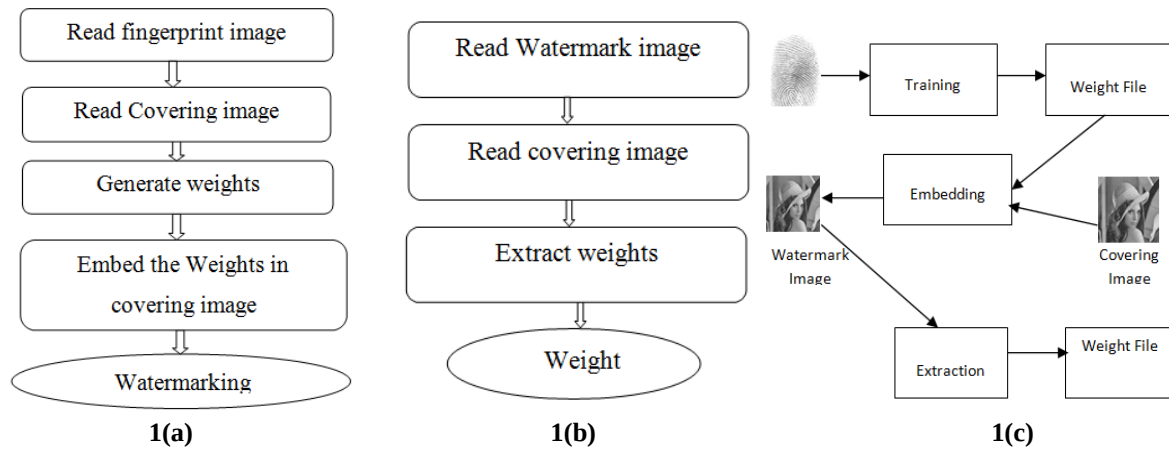


**Figure 1(a) Embedding Watermark    1(b)Extraction of weights    1(c) Weights Embedding and Extraction diagram**

### B. Extraction

- 1) The watermarked image is taken and the hidden neural network weights are extracted from the cover image by using a special technique described in algorithm below and the trained neural network is reconstructed.
- 2) The watermarked image is supplied at the input layer neurons and the final output watermark image is produced at the output layer.

### C. Algorithm For Weight Generation And Method For Embedding

Step 1: Let the target watermark image be  $timage$  the cover image used to produce the target watermark image be  $cimage$ .

Step 2: Convert  $timage$  as  $s$  a row vector

Step 3:  $cimage$  reshaped as a row vector

Step 4: A Back propagation algorithm based on a neural network with 1 input layer, 1 hidden layer and 1 output layer is used. The initial configuration of the back propagation network is chosen.

Let,

$n$  = Number of input layer neurons.

$m$  = Number of output layer neurons.

$h$  = Number of hidden layer neurons.

The weight matrix representing the weights connecting from input layer to hidden layer is represented by:

$v = \text{rand}(n, h) - 0.5$

The weight matrix representing the weights connecting the hidden layer neurons to the output layer is represented by:

$w = \text{rand}(h, m) - 0.5$

The initial bias of hidden layer neurons is set as:

$b1 = \text{rand}[1, h] - 0.5$

The initial bias of output layer neurons is set as :

$b2 = \text{rand}[1, m] - 0.5$

Let  $v1$  and  $w1$  are the matrices containing all zeros.  $v1$  and  $w1$  shall be used to record previous values of  $v$  and  $w$  matrix to calculate the momentum factor to speed up the learning process.

$v1 = \text{zeros}(n, h)$

$w1 = \text{zeros}(h, m)$

The learning rate is represented by  $\alpha$  and the momentum factor is represented by  $mf$ .

The controlling variable for the training of the image fragment  $con$  is initially set to 1.

$con = 1$

The total number of epochs to be used in training shall be stored in  $epoch$  and set to an initial value of 0.

$epoch = 0$

Now, the following section starts the training of the Back propagation Neural Network.

Step 5: Repeat the steps from 6 to 12 while  $con = 1$

Step 6: The error  $e$  is used to find difference between the target output and the output obtained and initialized to a value of 0.

$e = 0$  (13)

Step 7: Now to pick up each row of  $cimage$  for training, repeat the steps from 8 to 10 for each value of  $I$  from 1 to  $mc$ . (representing  $mc$  rows of the image section each with  $nc$  elements). Now, the output of the hidden layer and output layer neurons are calculated in the following steps.

Step 8: Let  $Z_{in}$  represents the net input to hidden layer neurons.  $Z_{in}$  is initialized with bias  $b_1$ .

$Z_{in}(j) = b_1(j)$  for  $1 \leq j \leq h$

The net input  $Z_{in}$  is calculated as:

$Z_{in}(j) = Z_{in}(j) + c_{image}(I,i) \times v(i,j)$  for  $1 \leq j \leq h$ ,

$1 \leq i \leq n$  (15)

The output of the hidden layer neurons is calculated by finding the binary sigmoid function of  $Z_{in}$ .

$Z(j) = \text{binsig}(Z_{in}(j))$

Let,  $Y_{in}$  represents the net input to the output layer.

$Y_{in}$  is initialized with a bias  $b_2$ .

$Y_{in}(k) = b_2(k)$  for  $1 \leq k \leq m$

The net input  $Y_{in}$  is calculated as:

$Y_{in}(k) = Y_{in}(k) + Z(j) \times w(j,k)$  for

$1 \leq j \leq h, 1 \leq k \leq m$

The output  $Y$  from the output layer neurons is given by:

$Y(k) = \text{binsig}(Y_{in}(k))$  for  $1 \leq k \leq m$

This output is stored in a matrix  $ty$ .

$ty(I,k) = Y(k)$  for  $1 \leq k \leq m$

Step 9: Now, the back propagation of error is done. The delta values at the output layer is given by:

$\text{delk}(k) = (t_{image}(I,k) - Y(k)) \times \text{binsigl}(Y_{in}(k))$  for  $1 \leq k \leq m$ ,

where,  $t_{image}(I,k) - Y(k)$  is the error at the  $k$ th neuron in the output layer.

The weights at the output layer are adjusted by :

$\text{delw}(j,k) = \alpha \times \text{delk}(k) \times z(j) + \text{mf} \times (w(j,k) - w_1(j,k))$  for  $1 \leq k \leq m, 1 \leq j \leq h$

The modifications in the bias of the output layer is calculated as:

$\text{delb2}(k) = \alpha \times \text{delk}(k)$ , for  $1 \leq k \leq m$

To calculate the delta values at the hidden layer, first,  $\text{delinj}$  is calculated and initialized to a value of 0.

$\text{delinj}(j) = 0$  for  $1 \leq j \leq h$

$\text{delinj}$  is modified with the help of  $\text{delk}$ .

$\text{delinj}(j) = \text{delinj}(j) + \text{delk}(k) \times w(j,k)$  for  $1 \leq k \leq m, 1 \leq j \leq h$

Now, delta value at the hidden layer neurons is calculated using  $\text{delinj}$ .

$\text{delj}(j) = \text{delinj}(j) \times \text{binsigl}(z_{in}(j))$ , for  $1 \leq j \leq h$

(This is used to calculate the modifications in the weight matrix  $v$ ).

The modifications in the weight matrix  $v$  is given by:

$\text{delv}[i,j] = \alpha \times \text{delj}[j] \times X[I,i] + \text{mf} \times (v[i,j] - v_1[i,j])$ , for  $1 \leq i \leq n, 1 \leq j \leq h$

The modifications in the biases of the input layer neurons is given by:

$\text{delb1}[j] = \alpha \times \text{delj}[j]$ , for  $1 \leq j \leq h$

Now, initial weights  $w$  and  $v$  are stored in  $w_1$  and  $v_1$  respectively. This is necessary to find the momentum factor during later stages to speed up training process.

$w_1[i,j] = w[i,j]$  for  $1 \leq i \leq n, 1 \leq j \leq m$  and  $v_1[i,j] = v[i,j]$  for  $1 \leq i \leq n, 1 \leq j \leq h$

Now, weight matrix  $w$  is updated.  $w[i,j] = w[i,j] + \text{delw}[i,j]$ , for  $1 \leq i \leq h, 1 \leq j \leq m$

The weight matrix  $v$  is updated.  $v[i,j] = v[i,j] + \text{delv}[i,j]$ , for  $1 \leq i \leq m, 1 \leq j \leq h$

The bias at the output layer is updated.

$b_2[k] = b_2[k] + \text{delb2}[k]$ , for  $1 \leq k \leq m$

The bias at the input layer is updated.

$b_1[j] = b_1[j] + \text{delb1}[j]$ , for  $1 \leq j \leq h$

The error  $e$  between the desired output and the output obtained is calculated by repeating equation 33 for each value of  $k$  from 1 to  $m$ .  $e = e + (t[I,k] - Y[k])^2$  for  $1 \leq k \leq m$

Step 10:  $I = I + 1$ , goto step 7 if  $I < mc$

Step 11: Modify the value of the controlling variable depending on total cumulative error  $e$  for the current image section.

If  $e < \text{min\_threshold\_error}$ ,  $\text{con} = 0$

Increment the current no. of epochs.

$\text{epochs} = \text{epochs} + 1$

Step 12: If  $\text{con} = 1$  then goto step 5, else follow step 13.

Step 13: Now, the trained Neural Network weights are hidden in the cover image using the following scheme.

Long format of double precision value has to be used for using this technique.

a) Pickup an image pixel intensity value of the given cover image and convert into string type.

b) At the end of the pixel value add '.0000'. Choose the weight value to be inserted after this. This does not affect the original pixel intensity value to a significant extent, as very least significant bits of the fractional part of the pixel value has been used.

c) Now, place the sign of the weight to be inserted as per following code. If weight value is negative, place 1, otherwise place 0, thus, the modified value becomes '.00001' of '.00000'

- d) Now, the numeric position just before the decimal point in the weight value is inserted. For. Ex. If the weight value is 256.78, the position just before the decimal point is 3. So, the string becomes '.000013'
- e) Now, the actual weight value excluding the decimal point is inserted. For ex. If the weight value is 256.78, we add 25678 at the end of the modified string.
- f) The final string becomes '.00001425678'
- g) This string is appended to the original pixel intensity value. For ex. If original pixel value is 234, it becomes '234.00001425678'. The weight value inserted into this pixel value is '-256.78'.
- h) Repeat steps from a) to g) for all pixel values of the cover image.

Next, the watermarked image with hidden trained network weights is supplied as input to the watermark extraction algorithm.

#### D. Extraction Algorithm

Step 1: The watermarked image is taken and the following scheme is used to extract the trained Neural Network weights from this image. The scheme works as given below.

- a) Get the modified pixel value from the watermarked image and convert into string type.
- b) If the string is in scientific format, convert into floating point format.
- c) Separate the original number and the embedded values as two separate strings. The string part before decimal point is the original pixel value and the string part starting from the decimal part is the embedded weight value. Thus, the original pixel value is obtained in this step.
- d) From the embedded part, the sixth character is used to decode the sign of the weight value. Inserted. If this value is 1, the weight value is negative, if it is 0 the weight value is positive.
- e) The seventh character of the embedded part is used to find the numerical position of the decimal point of the weight value.
- f) The embedded part from eight characters to the end of the string gives the weight value inserted without any decimal point. (value after removal of the decimal point from the weight value).
- g) Now, the actual decimal point is inserted into the value derived in (f) as per the position found in (e)
- h) Now, the original sign is inserted as derived in (d) and applied to the actual weight value. Thus the original weight value from the cover image pixel value is obtained.

Step 2: Repeat steps from a) to h) for all the pixels of the watermarked image.

Thus, the original weights matrices are extracted and reconstructed from the cover image. The derived weight matrices are used to reconstruct the Back propagation Neural Network.

### III. SAMPLE RESULTS

In this section we present the results for lena image as cover image and fingerprint image as the target image for which the neural network is trained to give weights. These weights are then embedded into the cover image. For the human eye the cover image and watermarked cover image are indifferent. At the extraction side the watermarked image is taken as input and weights are extracted from it. On comparing the embedded and extracted weights we find that a zero error function is obtained.

Fig 2(a) is a cover image and 2(b) is the target image which is used to train the neural network. The results with weights obtained after training as in fig 3(a) are shown embedded as in fig 3(b). The histograms of the cover image and watermarked images are shown in fig 4(a) and 4(b) respectively with a PSNR value of 43.5218dB.

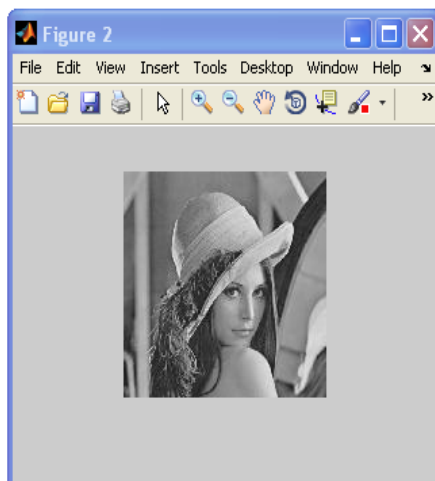
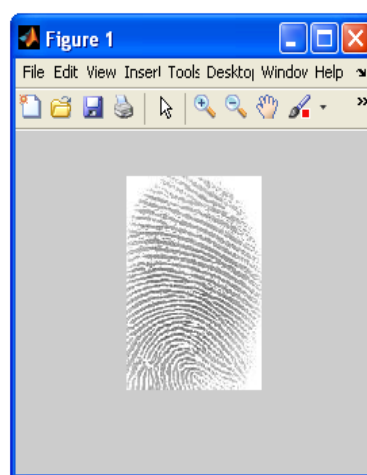


Figure 2.a Cover image



2.b Target image

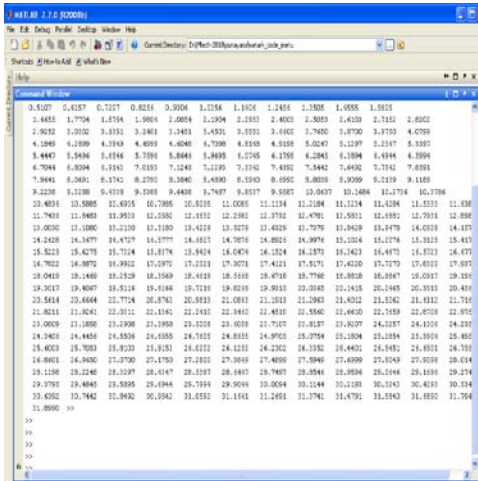


Figure 3(a). Weight generation process

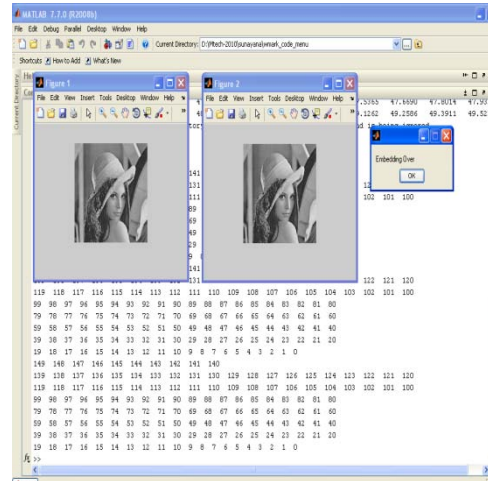


Figure 3(b). Embedding of weights for Lena Image

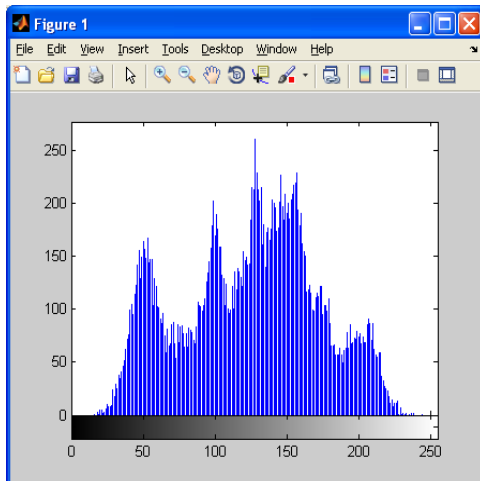
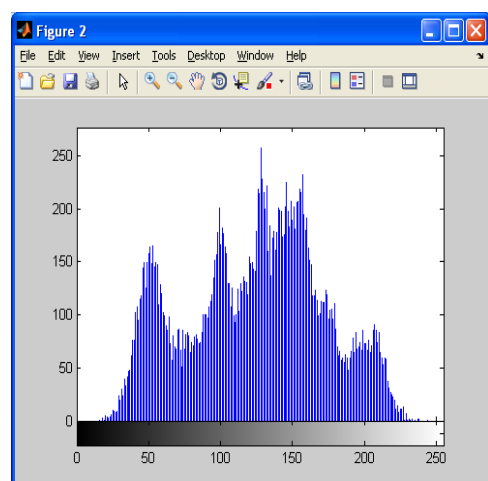


Figure 4(a). Histogram of original cover lena image



4(b) watermarked lena image(PSNR = 43.5218 dB)

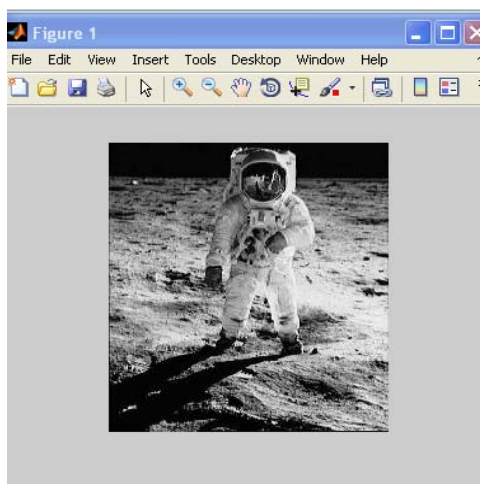
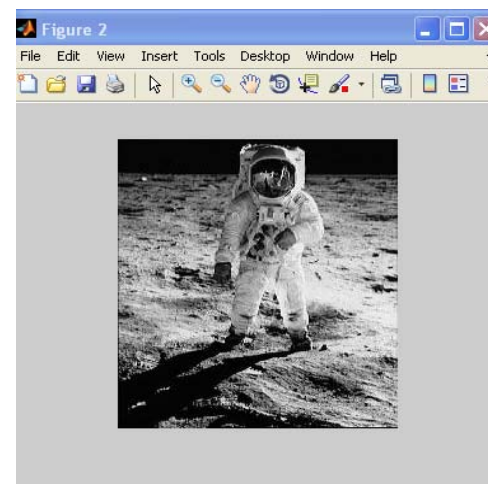


Figure 5(a) Cover Image Apollo



5(b) Watermarked Image Apollo (PSNR=47.2263dB)

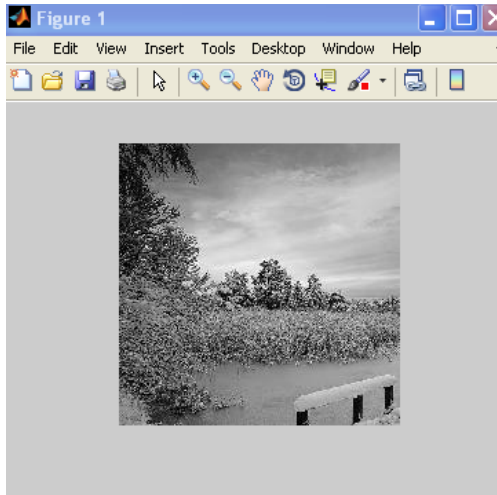
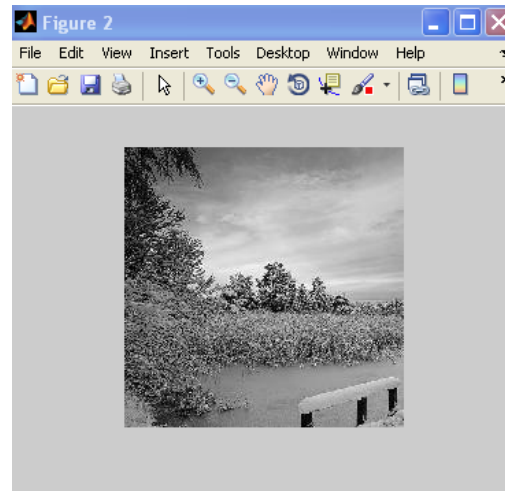


Figure 6(a) Cover Image Nature



6(b) Watermarked Image Nature( PSNR=46.2138dB)

Figure 5(a) and 6(a) are the cover images used to train the neural network for the same target image 2(b).The watermarked images are 5(b) and 6(b) with PSNR values 43.2263dB and 46.2138 dB respectively.

#### IV. Conclusion And Future Scope

This paper describes the algorithm to use a Back propagation Neural Network with an additional advantage of hiding the trained network weights within the original cover image. The watermarked image has a good robustness and imperceptibility of the cover image is also highly preserved as the average PSNR is 45dB .For the extraction, only cover image is required and no external weights files need to be supplied with the watermarked image. Thus, this work leads to a successful watermarking scheme. As the information was embedded in the weights of the neural network derived from the files, there was no visual deterioration of the watermark image obtained. This becomes particularly important as the technological disparity between individuals and organizations grows. Governments and businesses typically have access to more powerful systems and better encryption algorithms than individuals. Hence the chance of individual messages being broken increases with each passing year. Reducing the number of messages intercepted by the organizations as suspect will certainly help to improve privacy. Currently only limited finger prints are considered, this work can be extended for large database. In future this work can be applicable for hiding face, palm or signature trained files.

#### References

- [1] R.G.Van Schyndel, A.Z.Tirkel and C.F.Osborene, "A Digital Watermark" in Proc. IEEE International Conf. Image processing, 1994, vol.2 pp 86-92.
- [2] Ahmidi N. Safabakhsh R. "A Novel DCT Based Approach for Secure Color Image Watermarking" in Proc. ITCC 2004 International Conference Information Technology: Coding and computing, 2004, vol.2, pp 709-713.
- [3] K.J.Davis and K.Najarian "Maximizing Strength of Digital Watermarks Using Neural Networks", in Proc. International Joint Conf. Neural Network ,2001, vol 4, pp. 2893-2898. [4] Zhang Zhi Ming, Li Rong-Yan, Wang Lei, "Adaptive Watermark Scheme with RBF Neural Networks, in Proc. 2003 International Conf. Neural Networks and Signal Processing, 2003, vol 2. pp.1517-1520.
- [4] Ren -Junn Hwang, Chuan-Ho Kao and Rong- Chi Chang, "Watermark in Color Image" in Proc. First International symposium on cyber worlds, 2002 ,pp 225-229.
- [5] Fengsen Deng and Bingxi Wang, "A Novel Technique for Robust Image Watermarking in the DCT Domain" in Proc. Of the 2003 International Conf. Neural Networks and Signal Processing, 2003, vol.2, pp.1525-1528.
- [6] Fredric M.Ham and Ivica Kostanic, "Principles of Neurocomputing for Science & Engineering", Mc.GrawHill, Singapore, 2001, pp,136-140.
- [7] J.Cox, J.Kilian , "A Secure Robust Watermark for Multimedia" in Proc. First International Workshop, vol 1174 of Lecture notes in computer science, pp. 185-206.
- [8] R.Schyndel, A.Tirkel, and C.Osborne, "A Digital Watermark" in Proc. IEEE Int. Conf. on Image Processing, Nov. 1994 , Vol II, pp.86-90.