Application of Backpropagation Neural Network to generate fragmented watermarks and Full Watermark by Union

Er. Ashish Bansal[†], *Dr. Sarita Singh Bhadauria*[††], *Dr. Roopam Gupta*[†††]

[†]IT Department, Mahakal Institute Of Technology , 14-Azad Nagar, Ujjain, Madhya Pradesh, zip:456010, India,

[††]Electronics Department, Madhav Institute Of Technology and Science, Residency Road , Gwalior, Madhya Pradesh, zip:474005 ,India

[†††] IT Department, University Institute Of Technology, Bhopal, Madhya Pradesh, India

***Summary*—** Digital watermarking is a recent technology evolved to prevent illegal copy or reproduction of digital content. Most of the techniques developed use spatial and frequency domain for encoding the watermarks. These techniques fulfill the watermarking characteristics to varying degrees. There is a trade off observed between the information content and the fidelity of the cover image in almost all the works to a varying degree. This paper discusses a special scheme based on backpropagation neural network, which depends on small cover image parts to serve as inputs to a Backpropagation network and train them to produce corresponding small watermark image fragments. After training, the trained network weights are supplied with the cover image for the extraction of watermark. Small fragments of the cover image are taken to produce small fragments of the watermark image using trained weight matrix in the watermark extraction stage, which may be united to produce the original watermark image again. The watermark image is resistant to various image processing operations enhancing robustness of watermarking as the weights of the neural network remain unaffected by these operations.

***Key Words* —** Digital watermark, Neural Net, Backpropagation network.

I. Introduction

Digital watermarking should provide the qualities like imperceptibility, robustness and security of cover image. A large number of techniques have been developed based on manipulating the bit plane of Least Significant Bit (LSB)[1], linear addition of watermark to cover image[1], using mid band coefficients of DCT transformed blocks to hide watermark[2], maximizing strength of watermark using Discrete Wavelet Transform(DWT) techniques[3],using radial basis function(RBF)neural network to achieve maximum strength watermark[4], transforming color space of cover image and embedding watermark into saturation channel [5], Embedding watermark in the DC components of the transformed blocks [6] etc. Principles of neuro-computing, and their usage in science and technology is well explained in [7] .Cox et al. [8] pointed that, in order for a watermark to be robust to attack, it must be placed in perceptually significant areas of the image. Schyndel et al. [9] had generated a watermark using a m-sequence generator. Bas *et al*. [9] introduced a watermarking scheme using fractal codes. Bartolini et al. [10] utilized the properties of human visual system and generated watermark from DCT coefficients. Kundur and Hatzinakos [11] embedded the watermark in the wavelet domain where the strength of watermark was decided by the contrast sensitivity of the original image. Delaigle et el. [12] generated binary m-sequences and then modulated on a random carrier. A method for casting digital watermarks on images and analyzing its effectiveness was given by I.Pitas[13] and immunity to sub sampling was examined. Cox and Kilan [14] presented a secure algorithm for watermarking images using spread-spectrum techniques. Craver and Memon [15] proposed digital watermarks to resolve the copyright ownership. However, these techniques suffer from the problems of unsatisfactory value of imperceptibility and robustness to various attacks as discussed in these papers. These techniques also have the problems related to security.

The use of Neural Network for successful watermarking was effectively done in [16], where Full Counterpropagation Network (FCNN) was employed for the purpose of coding the cover image into a watermark image.

Chun –Yu-Chang [16] proposed a wonderful technique of embedding the watermarks into synapses of FCNN rather than cover image. This helped to increase robustness and reduce imperceptibility problems to a great extent. This paper is an attempt to explore Backpropagation Neural Network for Digital Watermarking applications. The cover image fragments are supplied as inputs to the input layer of Backpropagation Neural Network. The network is trained to produce fragments of the desired watermark. The robustness, fidelity and authenticity of the watermark generated is tested in later sections.

Manuscript received October 5, 2008
Manuscript revised October 20, 2008

Section II discusses the approach for using backpropagation Neural Network with fragmented cover image and fragmented target watermark image.
Section III provides the detailed algorithm for embedding and extraction. Section IV gives experimental results. Conclusion is given in Section V followed by references.

II. Approach For Using Backpropagation Neural Network With Fragmented Cover Image And Fragmented Target Watemark Image

The approach used for the proposed work is described as given below:

Embedding:

(1) The target watermark image is taken and divided into small fragments with 2 rows and 4 columns (2×4 size).
(2) The cover image is also fragmented into 2×4 parts.
(3) Backpropagation Neural Network is chosen with 1 input, 1 hidden and 1 output layer.
(4) The fragmented cover image parts are supplied as inputs to the input layer of the network respectively and weights are adjusted to produce the corresponding target image parts at the output layer.using Backpropagation algorithm.
(5) The trained network weights are stored in files.The cover image with the trained weights of the network is supplied for the purpose of extracting the watermark.

Extraction:

1) The watermarked image is taken, corrected by the image corrector network and fragmented into 2×4 parts.
2) The weights are extracted from the files and the trained neural network in the embedded stage is reconstructed.
3) The watermarked image parts are supplied at the input layer neurons and the target watermark fragments are produced at the output layer.
4) The output fragments so obtained are combined together to form the original complete output watermark image.

III. Algorithm

The following conventions apply to the embedding algorithm as well as extraction algorithms given below.

1) rand ('state', *s*) sets a random number generator to state *s*.
2) *s*= sign(*M*) generates a matrix s with same dimensions as matrix M and contains signs of the elements of matrix M.
s(*i*,*j*) = 1 , if *M*(*i*,*j*) > 0 for 1<=*i*<*mc*,1<=*j*<=*nc*
s(*i*,*j*) = -1, if *M*(*I*,*J*)<0 for 1<=*i*<*mc*,1<=*j*<=*nc*
where,
mc=Number of rows of matrix *M*.
nc= Number of columns of matrix *M*.
3) *M*= abs(*M*) generates a matrix *M* containing absolute values of the elements of *M*.
4) *M*= rand (*m*,*c*) generates a random matrix *M* containing *m* rows and *n* columns .
5) *M* =zeros (*m*,*c*) generates a matrix of *m* rows and *c* columns containing all zeros.
M (*i*,*j*) = 0 for 1<=*i*<=*m*,1<=*j*<=*c*.
6) *M* = binsig(*M*) generates a matrix containing binary sigmoid values of each value of the matrix *M*.
7) *M* = binsigl(*M*) generates: (write this)
8) *min_threshold_error* puts a lower bar on the acceptable value of error generated.
9) *M*=*M* UNION *N* appends the vector *N* at the end of the vector *M*.
10) *M*= reshape (*M*,*mc*,*nc*) reshapes the matrix *M* now with *mc* number of rows and *nc* number of columns.

A. EMBEDDING

Step 1: Let the target watermark image be given as:
$timage = [t_{11},t_{12},\ldots.t_{ij},\ldots\ldots t_{mc\times nc}]$
For $1<=i<=mc$, $1<=j<=nc$ (1)
Where, *mc*=number of rows in the target image.
And *nc*= number of columns in the target image.

And the cover image be given as :
$cimage=[c_{11},c_{12},\ldots.c_{ij},\ldots\ldots c_{mc\times nc}]$
For $1<=i<=mc$, $1<=j<=nc$ (2)
Where, *mc*=number of rows in the cover image.
And *nc*= number of columns in the cover image.

Step 2: The random number generator is initiated to original state *Rs* and a random matrix *ran* containing *mc* no. of rows and *nc* number of columns is generated with this state key. Setting the random number generator state is done by equation (3).
rand ('state',*Rs*)
ran= rand(*mc*,*nc*) (3)
$ran= [n_1,n_2,\ldots n_{ij},\ldots.n_{mc\times nc}]$ is now a matrix with *mc* rows and *nc* columns.

Step 3: The target image is normalized to contain pixel intensity values in the range from 0 and 1 for faster training and the original signs of the target image are stored in *org_sgn*.
timage(*i*,*j*) = *timage*(*i*,*j*)/255-*ran*(*i*,*j*)
For 1<=*i*<=*mc*, 1<=*j*<=*nc* (4)
org_sgn.=sign(*timage*) (5)
timage = abs(*timage*) (6)

Step 4: Now, *timage* is reshaped as a row vector containing

$mc \times nc$ number of columns.

$timage[(i\text{-}1) \times nc+j]=t[i,j]$ for $1<=i<=mc$, $1<=j<=nc$ (7)

This produces a row vector $timage[t_1,t_2,\ldots.t_{mc \times nc}]$.

Step 5: Let the target watermark image be fragmented into L number of segments given by:

$L=(mc \times nc)/8$ (8)

(Each segment in L contains 8 elements).

Step 6: Let the remaining number of values in the target image matrix after extracting all 8 segments is termed as *padlength* given by

$padlength = mc \times nc - L \times 8$ (9)

Step 7: A Backpropagation algorithm based on a neural network with 1 input layer, 1 hidden layer and 1 output layer is used.

Let f denotes the fragment location.

$f=0$ (10)

Step 8: Now, we start a loop to pickup each segment of the target image.

For each value of *outloop* from 1 to L repeat the steps from 9 to 22.

Step 9: Pickup image fragments.

The starting location of the image fragment f is given as

$f=outloop \times 8\text{-}7$ (11)

Pick up image fragment t containing 8 elements.

$t = timage[f,f+1,f+2, \ldots., f+7]$ (12)

Step 10: Reshape the image fragment t into a two dimension matrix containing 2 rows and 4 columns.

$t(1,j) = t(j)$ for $1<=j<=4$ (13)

$t(2,j\text{-}4) = t(j)$ for $5<=j<=8$ (14)

The equations 13 and 14 produce a two dimension matrix t with 2 rows and 4 columns..

Step 11:

Pick up image fragment c containing 8 elements.

$c = cimage[f,f+1,f+2, \ldots., f+7]$ (15)

Reshape the cover image fragment c into a two dimension matrix containing 2 rows and 4 columns.

$c(1,j)= c(j)$ for $1<=j<=4$ (16)

$c(2,j\text{-}4) = c(j)$ for $5<=j<=8$ (17)

Let X denotes the selected 2*4 part of the cover image.

$X=c$ (18)

Step 12: Now, the initial configuration of the backpropagation 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:

$v=rand(n,h) - 0.5$ (19)

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

$w= rand(h,m) - 0.5$ (20)

The initial bias of hidden layer neurons is set as:

$b1 = rand[1,h] - 0.5$ (21)

The initial bias of output layer neurons is set as:

$b2=rand[1,m]- 0.5$ (22)

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 before updation during each cycle of training to calculate the momentum factor to speed up the learning process.

$v1= zeros(n,h)$ (23)

$w1= zeros(h,m)$ (24)

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.

$con= 1$ (25)

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

$epoch = 0$ (26)

Now, the training starts with the image section t as the target output and random matrix X as random input matrix.

Step 13:

Repeat the steps from 14 to 20 while *con*=1

Step 14: The error e is used to find difference between the target output and the output obtained and initialized to a value of 0.

$e=0$ (27)

Step 15: Now to pick up each row of X for training, repeat the steps from 16 to 18 for each value of I from 1 to 2.(representing 2 rows of the image section each with 4 elements).

Now, the output of the hidden layer and output layer neurons

are calculated in the following steps.

Step 16: Let *Zin* represents the net input to hidden layer neurons.
Zin is initialized with bias *b*1.
$Zin(j) = b1(j)$ for $1<=j<=h$ (28)
The net input *Zin* is calculated as:
$Zin(j) = Zin(j) + X(I,i) \times v(i,j)$ for $1<=j<=h$, $1<=i<=n$ (29)
The output of the hidden layer neurons is calculated by finding the binary sigmoid function of *Zin*.
$Z(j) = \text{binsig}(Zin(j))$ for $1<=j<=h$ (30)
Let, *Yin* represents the net input to the output layer.
Yin is initialized with a bias b2.
$Yin(k) = b2(k)$ for $1<=k<=m$ (31)
The net input *Yin* is calculated as:
$Yin(k) = Yin(k) + Z(j) \times w(j,k)$ for $1<=j<=h$,$1<=k<=m$ (32)
The output *Y* from the output layer neurons is given by:
$Y(k) = \text{binsig}(Yin(k))$ for $1<=k<=m$ (33)
This output is stored in a matrix *ty*.
$ty(I,k) = Y(k)$ for $1<=k<=m$ (34)

Step 17: Now, the backpropagation of error is done.

The delta values to calculate weight adjustments at the output layer are given by:
$delk(k) = (t(I,k) - Y(k)) \times \text{binsigl}(Yin(k))$ for $1<=k<=m$,
Where, $t(I,k) - Y(k)$ is the error at the *k*th neuron in the output layer.
The weights at the output layer are adjusted by:
$delw(j,k) = alpha \times delk(k) \times z(j) + mf \times (w(j,k)\text{-}w1(j,k))$
For $1<=k<=m$, $1<=j<=h$ (35)

The modifications in the bias of the output layer is calculated as: $delb2(k)=alpha \times delk(k)$, for $1<=k<=m$ (36)

To calculate the delta values to calculate the weight adjustments at the hidden layer,
First, *delinj* is calculated and initialized to a value of 0.
$delinj(j) = 0$ for $1<=j<=h$ (37)

delinj is modified with the help of *delk*.

$delinj(j) = delinj(j)+delk(k) \times w(j,k)$ for $1<=k<=m$,
$1<=j<=h$ (38)

Now, delta value at the hidden layer neurons *delj* is calculated using *delinj*.

$delj(j)= delinj[j] \times \text{binsigl}(zin[j])$, for $1<=j<=h$ (39)
(This is used to calculate the modifications in the weight matrix *v*).

The modifications in the weight matrix *v* are given by:
$delv[i,j] = alpha \times delj[j] \times X[I,i] + mf \times (v[i,j]\text{-}v1[i,j])$,
For $1<=i<=n$, $1<=j<=h$ (40)

The modifications in the biases of the input layer neurons is given by: $delb1[j] = alpha \times delj[j]$, for $1<=j<=h$ (41)

Now, previous 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.

$w1[i,j]=w[i,j]$ for $1<=i<=h$, $1<=j<=m$
and
$v1[i,j]=v[i,j]$ for $1<=i<=n$, $1<=j<=h$ (42)

Now, weight matrix *w* is updated.
$w[i,j]= w[i,j]+delw[i,j]$, for $1<=i<=h$, $1<=j<=m$ (43)
The weight matrix *v* is updated.
$v[i,j] = v[i,j] + delv[i,j]$, for $1<=i<=n$, $1<=j<=h$ (44)
The bias at the output layer is updated.
$b2[k] = b2[k]+ delb2[k]$, for $1<=k<=m$ (45)
The bias at the input layer is updated.
$b1[j] = b1[j] + delb1[j]$, for $1<=j<=h$ (46)

The error *e* between the desired output and the output obtained is calculated by repeating equation 43 for each value of *k* from 1 to *m*.

$e=e+ (t[I,k]\text{-}Y[k])\text{^}2$ for $1<=k<=m$ (47)

Step 18: $I=I+1$, goto step 15 if $I<3$ (48)

Step 19: Modify the value of the controlling variable depending on total cumulative error *e* for the current image section.

If $e<min\_threshold\_error$
then con= 0 (49)
Increment the current no. of epochs.
$epoch = epoch+1$ (50)

Step 20: If *con*=1 then go to step 13, else follow step 21.

Step 21: Now, the trained weight matrices are stored in files.

The files *wfile*, *vfile*, *b1file*, *b2file* are opened in “write” mode.

The weight matrix *w* is stored in *wfile*.

The weight matrix *v* is stored in *vfile*.
The bias matrix *b*1 is stored in *b*1*file*.
The bias matrix *b*2 is stored in *b*2*file*

Step 22: Go to step 8 if *outloop*<=*L*

Step 22:

Now, all the files are closed.
The random state *Rs* is stored in higher precision bit of the fractional value of the pixel value intensity.
Now, this watermarked image is supplied with the trained weight matrix files for the watermark extraction algorithm.
.

B. EXTRACTION

Step 1: The initial random state key *Rs* is extracted from the higher precision bit of the fractional part of the pixel value intensity of the image and the random number generator is set to state specified by the random state key *Rs*.

Step 2: Open all the files *wfile*, *vfile*, *b1file*, *b2file* in read mode to read the trained weight matrices of Backpropagation network corresponding to each image fragment respectively.

Step 3:
Now, for each value of *outloop* from 1 to *L* repeat the steps from 4 to 15. (This is to pickup each image fragment).

Step 4: Pickup the image section *X* as indicated in the Embedding algorithm.

Step 5: Read from the trained weight files and populate the corresponding weight matrices. (Corresponding to the successive image sections.. one by one.)

w is populated from *wfile*.
v is populated from *vfile*.
*b*1 is populated from *b1file*.
*b*2 is populated from *b2file*.

Step 6: For each value of *I* from 1 to 2, perform the steps from
7 to 14

Step 7: Initialize *Zin* with the bias *b*1.
$Zin(j)= b1(j)$, for $1<=j<=h$ (51)

Step 8: Find the net input *Zin* to hidden layer neurons.
$Zin(j)= Zin(j) + X(I,i) \times v(I,j)$, for $1<=i<=n$, $1<=j<=h$ (52)

Step 9: The output of the hidden layer neuron is calculated as:
$Z(j) = \text{binsig}(Zin(j))$, for $1<=j<=h$ (53)

Step 10: Initialise *Yin* with the bias *b*2.
$Yin[k]= b2[k]$ for $1<=k<=m$ (54)

Step 11: Now, the net input to the output layer neuron *Yin* is calculated as:
$Yin[k] = Yin[k] + Z[j] \times w[j,k]$, for $1<=j<=h, 1<=k<=m$ (55)

Step 12: The output from the output layer neuron is calculated as: $Y[k] = \text{binsig}(Yin[k])$, for $1<=k<=m$ (56)

Step 13: This output is stored in *ty*.
$ty[I,k] = Y[k]$, for $1<=k<=m$ (57)

Open a file *tyfile* in "write" mode to and store *ty* in *tyfile*.

Step 14: *I*=*I*+1
If I<2 goto step 6 else goto step 15.
Step 15: *outloop* = *outloop* +1
Goto step 3 if *outloop*<*L*+1 else goto step 16.

Step 16: Close all files.
Now, open the *tyfile* in "read" mode.
Now, each image fragment is read from *tyfile* and appended to a matrix *obt_image*.

Step 17: Initialize obt_image as a null matrix.
obt_image = [] (58)

Step 18: Now read all image sections from *tyfile* into *ty* matrix. For each value of *outloop* from 1 to *L* repeat steps from 19 to 22.

Step 19: Read image section from the *tyfile* into matrix *ty* of dimension 2×4. (2 rows and 4 columns).

Step 20: Now *ty* is reshaped into a row vector of dimension (1×8).
$ty[(i-1) \times 4+j] = t[i,j]$ for $1<=i<=2$, $1<=j<=4$ (59)
This provides a row vector *ty*= [*ty*1, *ty*2,….*ty*8]

Step 21: Now, the row vector *ty* is appended to *obt_image*.
obt_image= *obt_image* U *ty* (60)

Step 22: *outloop* = *outloop*+1
Goto step 18 if outloop<L+1 else go to step 23.

Step 23: Now, a vector of zeros of length equal to *padlength* is appended to the *obt_image*.

padder = zeros(1,*padlength*) (61)

obt_image=*obt_image* U *padder* (62)

Step 24: Now, reshape the *obt_image* into a matrix with *mc* number of rows and *nc* number of columns.

obt_image[1,*j*]= *obt_image*[*j*], for 1<=*j*<=*nc*, *i*=1 (63)

obt_image[*i*,*j*-4*(*i*-1)]= *obt_image*[*j*],for (*nc*+*i*-1)<=*j*<=*nc*,*i*>1 (64)

Step 25: Now to set the random state generate the original random state *Rs* to calculate the random matrix *ran*.

rand ('state',*Rs*)

ran= rand(*mc*,*nc*) (65)

Step 26: Now, the original signs from *org_sign* are included into *obt_image*.

obt_image(*i*,*j*)=*obt_image*(*i*,*j*) × *org_sign(i,j)*

For 1<=*i*<=*mc*, 1<=*j*<=*nc* (66)

Step 27: Now, to reverse the effect of normalization done in step 3 during embedding.

obt_image[*i*,*j*]=*obt_image*[*i*,*j*]+*ran*[*i*,*j*], for 1<=*i*<=*mc*, 1<=*j*<=*nc* (67)

And

obt_image[*i*,*j*] = *obt_image*[*i*,*j*] × 255, for 1<=*i*<=*mc*, 1<=*j*<=*nc* (68)

Step 28: Now *obt_image* is displayed.

This is the target watermark image reconstructed from the watermarked image fragments and available as network output.

IV. Experiments conducted WITH AND THE results:

a) In this experiment, variation of PSNR is shown with respect to change in threshold value. The threshold is varied from 0.4 to 0.0001 as shown in table – I. With the reduction in the threshold value, the PSNR goes on increasing. There is also an increment seen in training time and number of epochs required for training. The values of α is kept at 4 and the value of *mf* is also kept constant at 0.8. The PSNR varies from 16.21 to 40.66. The best PSNR value is obtained at threshold value of 0.0001 with a training time of 322.85 seconds and number of epochs as 223499. Fig. 1 to Fig. 4 show the extracted watermark image corresponding to threshold values of 0.1, 0.01, 0.001 and 0.0001 respectively.

TABLE- I
(Variation of PSNR with threshold)
(α =4, *mf*= 0.8)

α	mf	Threshold	PSNR	Training time	Epochs
4	0.8	0.4	16.21	18.91	6445
4	0.8	0.3	17.28	21.42	8412
4	0.8	0.2	18.81	24.79	11345
4	0.8	0.1	20.89	32.20	16900
4	0.8	0.01	29.72	66.71	38835
4	0.8	0.001	35.73	128.43	83438
4	0.8	0.0001	40.66	322.85	223499

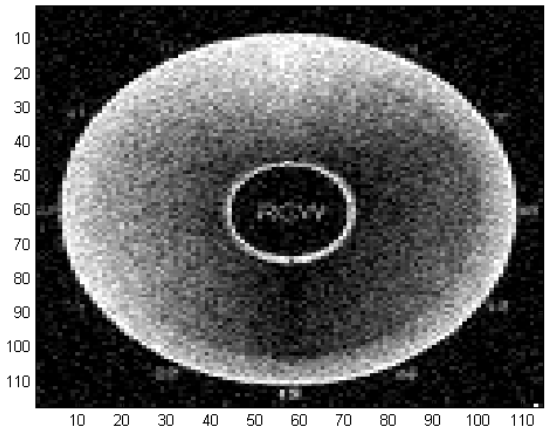

FIG 1 (Threshold=0.1)

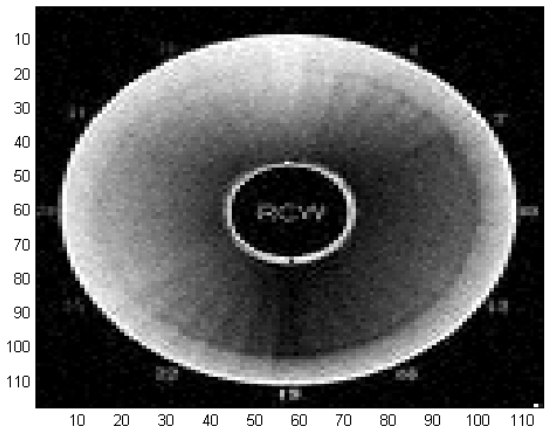

FIG.2 (Threshold = 0.01)

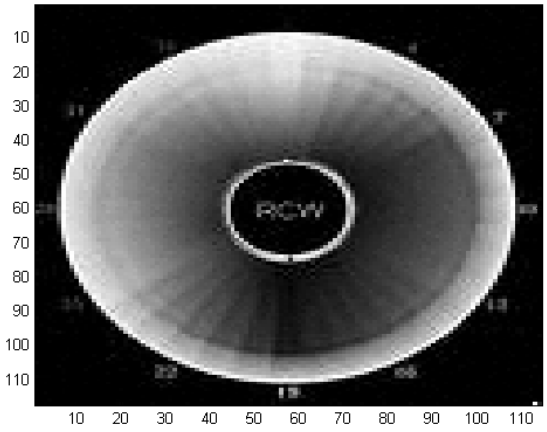

FIG. 3 (Threshold = 0.001)

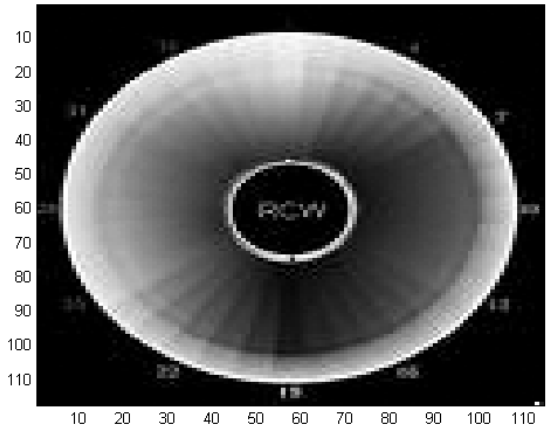

FIG. 4 (Threshold = 0.0001)

b) In the second experiment, the variation of training time with respect to *mf* is seen. m*f* is used to accelerate the training time and pushing the training in the direction of previous weight changes. For this experiment, α is kept at a constant value of 0.9 and *threshold* is kept constant at 0.5. The training time reduces with the increase in *mf* from 0.1 to 0.9 in steps of 0.1. The training time varies from 41.96 sec. to 22.40 seconds and the number of epochs used to train the network reduces from 22800 to 9126 with the variation in *mf* from 0.1 to 0.9 respectively. The chart presented in Fig. 5 represents the variation in training time with respect to *mf*. The reducing trend of training time with increase in *mf* is clearly visible from this chart.

TABLE – II
(Variation of Training time with mf)
(α =0.9, *threshold*=0.5)

Threshold	mf	α	Training time	Epochs
0.5	0.1	0.9	41.96	22800
0.5	0.2	0.9	38.77	22780
0.5	0.3	0.9	34.14	16547
0.5	0.4	0.9	28.88	14003
0.5	0.5	0.9	25.95	11591
0.5	0.6	0.9	22.29	9683
0.5	0.7	0.9	20.99	8219
0.5	0.8	0.9	20.23	7599
0.5	0.9	0.9	22.40	9126

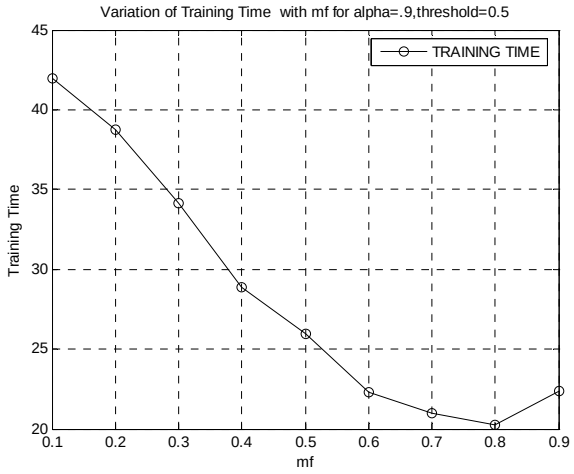

Fig. 5 (Variation of training time with *mf*)

c) In the third experiment, the variation of training time is plotted with respect to α, keeping threshold constant at 0.5 and *mf* constant at 0.3. α represents the learning rate of the neural network. With the reduction in α, the training time and number of epochs increase gradually. With the variation of α from 4 to 0.4, the training time increases from 22.50 sec. to 53.26 sec. and the epochs used to train the network increase from 10038 to 29668 respectively. Fig. 8 shows the chart showing the variation of training time with α. The reduction in training time with increased value of α is clearly seen from the chart. Fig. 9 shows the chart showing the reduction in number of training epochs required with increase in the value of α.

TABLE – III
(Variation of Training time with α)
(Threshold=0.5, mf=0.3)

Threshold	mf	α	Training time	Epochs
0.5	0.3	4	22.50	10038
0.5	0.3	3	25.00	10877
0.5	0.3	2	25.78	11484
0.5	0.3	1	32.44	15669
0.5	0.3	0.8	36.00	17855
0.5	0.3	0.7	38.33	19654
0.5	0.3	0.6	38.92	21862
0.5	0.3	0.5	46.97	24955
0.5	0.3	0.4	53.26	29668
0.5	0.3	0.3	64.21	37456
0.5	0.3	0.2	86.22	51992

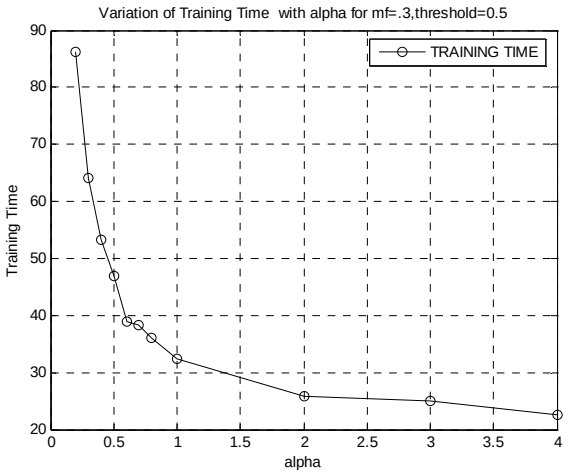

FIG.6 (Training time v/s alpha)

d) In this experiment, the robustness of the watermarking scheme is shown. The cover image of Lena shown in fig. 7 contains only the random generator key embedded in the higher precision bits of the intensity value of the first pixel of the image. As the various attacks failed to disturb this key and the information was embedded in the weights of the neural network derived from the files and there is an image corrector applied before extracting the final watermark which restores the fidelity of the un-attacked image, there was no visual deterioration of the watermark image obtained. The various attacks used were blurring, cropping, and sharpening, rotation, and scaling and JPEG compression. In each case, PSNR value of the watermark image was obtained for the threshold values varying from 0.1 to 0.0001 respectively. The table IV shows the obtained values of PSNR for each of these attacks. It is seen that these values are exactly same as shown in table I. This is possible only because the watermarked image of Lena does not contain the actual information. In fact, the actual information is derived from the weights of the neural network already saved in files during the training step. Only, the random state key was embedded in the cover image of Lena and it was also saved with the files. This key is being embedded in the cover image mainly for the purpose of authentication as discussed in later experiments.

TABLE – IV

Attack	Threshold values			
	0.1	0.01	0.001	0.0001
	(PSNR)	(PSNR)	(PSNR)	(PSNR)
Blurred	20.89	29.72	35.73	40.66
Cropped	20.89	29.72	35.73	40.66
Sharpen	20.89	29.72	35.73	40.66
Rotation	20.89	29.72	35.73	40.66
Scaling	20.89	29.72	35.73	40.66
JPEG	20.89	29.72	35.73	40.66

e) In this experiment, the test of imperceptibility is done. The cover image taken was Lena's image. The random state key was hidden in the higher precision of first pixel value. Let the cover image (Lena's image) is given as:
$Y= [y_{11}, y_{12},\ldots.y_{ij},\ldots\ldots y_{mc\times nc}]$ (Fig. 8)
The first pixel value Y (1, 1) = 136 is changed to
Y (1,1) = 136.0010. (Fig. 7)
The last 2 bits represent the hidden state key. This value is extracted and used testing authenticity of the watermarked image. The PSNR value of the watermarked image of Lena after the insertion of the random state key, with respect to the original picture of Lena is calculated as 145.5371.This high value of PSNR indicates that, there is a very little deterioration in the quality of cover image by insertion of the random state key. Thus, the property of imperceptibility is highly preserved under this scheme

FIG.7 (Lena's cover image containing random generator key)

FIG. 8 (Original Lena's image)

when only a small state key is inserted .However, if the encoding matrix of 4*8 size is inserted by DCT encoding method, the PSNR of watermarked image comes to 48.68.

This is because the payload inside the image increases and thus reduces the fidelity. In practical situations, there is a likelihood of having bigger payload for the sake of authentication**.**

f) Authenticity test:

The random state key is derived from the high precision bits of the first pixel of the cover image of Lena and then compared with the value of random state key stored in the file during the algorithm. If the two values match, authenticity is preserved, otherwise, the authenticity is suspected. Thus, authenticity feature is also well preserved in this scheme. Also, when random numbers are generated with a wrong key of 1000, the output obtained is not as expected .Thus, authenticity is preserved.

Conclusions:

In this paper, Backpropagation Neural Network is being used for training cover image fragments into corresponding target watermark image fragments. Encoding the watermark using the weights of Backpropagation network has reduced the chances of watermark destruction with image processing operations. The watermark image needs to be supplied with the trained network weights to produce the watermark output. As the target watermark image is normalized with the help of random state key *Rs* which is stored in the image itself, this is used for the authenticity purpose. Results have revealed that a Backpropagation Neural Network may be successfully employed to provide a successful watermarking scheme by training the cover image fragments for the corresponding target watermark image fragments.

REFERENCES

[1] R.G.Van Schyndel,A. Z.Tirkel and CF. Osborene, "A Digital Watermark" in Proc. IEEE International Conf. Image processing,1994,vol.2 pp 86-92.

[2] Ahmidi N. Safabaksh 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.

[5] Ren –Junn Hwand,Chuan-Ho Kao and Rong-Chi Chang, "Watermark in Color Image" in Proc. First International symposium on cyber worlds, 2002, pp 225-229.

[6] 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.

[7] Fredric M.Ham and Ivica Kostanic, "Principles of Neurocomputing for Science & Engineering", Mc.GrawHill, Singapore, 2001, pp, 136-140.

[8] 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.

[9] 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.

[10] F.Bartolini,M.Barni,V.Cappellini ad A.Piva, "Mask Building for Perceptually Hiding Frequency Embedded Watermarks", in Proc. Int.Conference on Image Processing ,Oct. 1998,vol. I, pp. 450-454.

[11] D.Kundur and D. Hatzinakos, "A Robust Digital Image Watermarking Method using Wavelet – Based Fusion", in Proc, IEEE Int. Conf. on Image Processing , Oct. 1997, vol. I, pp. 544-547.

[12] J.Delaigle,C.De Vleeschouwer, and B. Macq, "Psychovisual Approach to Digital Picture Watermarking", Journal of Electronic Imaging, vol.7,No.3,pp.628-640,July 1998.

[13] I.Pitas, "A Method for Signature Casting on Digital Images", in Proc, IEEE Int. Conf. on Image Processing ,Sept 1996,vol.III,pp.215-218.

[14] I.Cox,J Kilan,"Secure Spread Spectrum Watermarking for Images, Audio and Video" , in Proc. IEEE International Conference on Image Processing ,1996,vol 3,pp. 243-246.

[15] S. Craver, N. Memon, "Resolving Rightful Ownership with Invisible Watermarking Techniques: Limitations, Attacks and Implications", IEEE Trans., vol 16, No. 4,pp. 573-586,1998.

[16] Chun-Yu-Chang, "The Application of a Full Counterpropagation Neural Network to Image Watermarking", 2005, IEEE