WATERMARKING USING NEURAL NETWORK AND HIDING THE TRAINED NETWORK WITHIN THE COVER IMAGE

[1] Er. Ashish Bansal , [2] Dr. Sarita Singh Bhadauria
[1]Reader,Department of Information Technology,MIT,Ujjain,India
[2]Professor,Department of Electronics,MITS,Gwalior,India
E-mail: ashssi@rediffmail.com , saritamits61@yahoo.co.in

ABSTRACT

The major source of communication in present world is digital media. As it is quite easy to manipulate a digital media, it becomes essential to protect the digital information by legitimate means. Digital Watermarking has evolved as one of the latest 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. However, there is always a tradeoff between robustness and imperceptibility features of watermarking offered by these techniques. This paper offers a technique based on Backpropagation 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 has 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 again which in turn converts the cover image into desired watermark image. 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.

Keywords : *Digital Watermarking, Information Hiding, Digital Media Protection, Neural Network*

1. INTRODUCTION

Digital watermarking should provide the qualities like imperceptibility, robustness, 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 transformed blocks[6] etc. Principles of neurocomputing, 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] 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 subsampling 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 used to insert the watermark into synapses of FCNN rather than the cover 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 has described the use of a Backpropagation 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.

Section 2 discusses the approach for using Backpropagation with the help of broad outline of the technique of embedding and extraction.

Section 3 provides the detailed algorithm for embedding and extraction . Section IV gives experimental results . Conclusion is given in Section 4 followed by references.

2. APPROACH FOR USING BACKPROPAGATION NEURAL NETWORK WITH COVER IMAGE AND GIVEN TARGET WATERMARK IMAGE

The approach followed for the proposed work is described as follows:

2.1 Embedding:

1) The target watermark image is taken to serve as output to a Backpropagation Neural Network.
2) A Backpropagation 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 Backpropagation algorithm.
4) The trained weight matrix is hidden within the cover image itself by using special technique described in algorithm given in section III.

2.2 Extraction:

1) The watermarked image is taken and the hidden neural network weights are extracted from the cover image by using a special techniquedescribed in algorithm given in section III 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.
3) The output watermark image is correlated with the target output watermark to determined PSNR of the obtained watermark image.

3. ALGORITHM

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

1) M= rand(m,c) generates a random matrix M containing m rows and n columns .
2) M =zeros(m,c) generates a matrix of m rows and c columns containing all zeros.
$M(i,j) = 0$ for $1<=i<=m, 1<=j<=c$.
3)M = binsig(M) generates a matrix containing binary sigmoid values of each value of the matrix M.
4)M = binsigl(M) generates:
binsig(M)(1-binsig(M))
5) *min_threshold_error* puts a lower bar on the acceptable value of error generated.

3.1 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.

The cover image used to produce the target watermark 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: Now, *timage* is reshaped as a row vector containing $mc\times nc$ number of columns.
$timage[(i-1)\times nc+j]=t[i,j]$ for $1<=i<=mc$, $1<=j<=nc$ (3)
This produces a row vector $timage[t_1,t_2,\ldots.t_{mc\times nc}]$.

Step 3: Now, c*image* is reshaped as a row vector containing $mc \times nc$ number of columns.

$cimage[(i\text{-}1) \times nc+j]=c[i,j]$ for $1<=i<=mc$, $1<=j<=nc$ (4)

This produces a row vector $cimage[c_1,c_2,\ldots.c_{mc \times nc}]$.

Step 4: A Backpropagation algorithm based on a neural network with 1 input layer, 1 hidden layer and 1 output layer is used. 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 (5)

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

w= rand(h,m) – 0.5 (6)

The initial bias of hidden layer neurons is set as:

$b1$ = rand[1,h] – 0.5 (7)

The initial bias of output layer neurons is set as :

$b2$=rand[1,m]- 0.5 (8)

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.

$v1$= zeros(n,h) (9)

$w1$= zeros(h,m) (10)

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 (11)

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 (12)

Now, the following section starts the training of the Backpropagation Neural Network.

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

Step 6: The error *e* is 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 *Zin* represents the net input to hidden layer neurons.

Zin is initialized with bias $b1$.

$Zin(j) = b1(j)$ for $1<=j<= h$ (14)

The net input *Zin* is calculated as:

$Zin(j) = Zin(j) + cimage(I,i) \times v(i,j)$ for $1<=j<=h$, $1<=i<=n$ (15)

The output of the hidden layer neurons is calculated by finding the binary sigmoid function of *Zin*.

$Z(j) = \text{binsig}(Zin(j))$ (16)

Let, *Yin* represents the net input to the output layer.

Yin is initialized with a bias $b2$.

$Yin(k) = b2(k)$ for $1<=k<=m$ (17)

The net input Yin is calculated as:

$Yin(k) = Yin(k) + Z(j) \times w(j,k)$ for $1<=j<=h$,$1<=k<=m$ (18)

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

$Y(k) = \text{binsig}(Yin(k))$ for $1<=k<=m$ (19)

This output is stored in a matrix *ty*.

$ty(I,k) = Y(k)$ for $1<=k<=m$ (20)

Step 9: Now, the backpropagation of error is done.

The delta values at the output layer is given by:

$delk(k)= (timage(I,k) – Y(k)) \times \text{binsigl}(Yin(k))$ for $1<=k<=m$,

where, $timage(I,k) – Y(k)$ is the error at the kth 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 $1<=k<=m$, $1<=j<=h$ (21)

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

$1<=k<=m$ (22)

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

$delinj(j) = 0$ for $1<=j<=h$ (23)

delinj is modified with the help of *delk*.

$$delinj(j) = delinj(j)+delk(k) \times w(j,k) \text{ for } 1<=k<=m,\ 1<=j<=h \quad (24)$$

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

$$delj(j)= delinj[j] \times \text{binsigl}(zin[j]),\ \text{for } 1<=j<=h \quad (25)$$

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

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

$$delv[i,j] = alpha \times delj[j] \times X[I,i] + mf \times (v[i,j]-v1[i,j]),\ \text{for } 1<=i<=n,\ 1<=j<=h \quad (26)$$

The modifications in the biases of the input layer neurons is given by: $delb1[j] = alpha \times delj[j]$, for $1<=j<=h$ (27)

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.

$$w1[i,j]=w[i,j] \text{ for } 1<=i<=n,\ 1<=j<=m$$
and
$$v1[i,j]=v[i,j] \text{ for } 1<=i<=n,\ 1<=j<=h \quad (28)$$

Now, weight matrix w is updated.

$$w[i,j]= w[i,j]+delw[i,j] \text{ , for } 1<=i<=h,\ 1<=j<=m \quad (29)$$

The weight matrix *v* is updated.

$$v[i,j] = v[i,j] + delv[i,j], \text{ for } 1<=i<=m,\ 1<=j<=h \quad (30)$$

The bias at the output layer is updated.

$$b2[k] = b2[k]+ delb2[k], \text{ for } 1<=k<=m \quad (31)$$

The bias at the input layer is updated.

$$b1[j] = b1[j] + delb1[j], \text{ for } 1<=j<=h \quad (32)$$

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*.

$$e=e+(t[I,k]-Y[k])\text{^}2 \text{ for } 1<=k<=m \quad (33)$$

Step 10: $I=I+1$, goto step 7 if $I< mc$ (34)

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

If $e<min\_threshold\_error$, $con= 0$ (35)

Increment the current no. of epochs.

$$epochs=epochs+1 \quad (36)$$

Step 12: If *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 if 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.

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

.

3.2 Extraction

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 character 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.

i) Repeat steps from to 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 Backpropagation Neural Network to be used in later steps of the algorithm. In the process, the original cover image is also obtained in step c) described above..

Step 2: For each value of I from 1 to *mc*, perform the steps from 3 to 10.

Step 3: Initialize *Zin* with the bias *b*1.

$$Zin(j) = b1(j), \text{ for } 1<=j<=h \quad (37)$$

Step 4: Find the input *Zin* to hidden layer neurons.

$$Zin(j) = Zin(j) + cimage(I,I) \times v(I,j), \text{ for } 1<=i<=n, 1<=j<=h \quad (38)$$

Step 5: The output of the hidden layer neuron is calculated as:

$$Z(j) = \text{binsig}(Zin(j)), \text{ for } 1<=j<=h \quad (39)$$

Step 6:Initialise *Yin* with the bias *b*2.

$$Yin[k]=b2[k] \text{ for } 1<=k<=m \quad (40)$$

Step 7: Now, the net input to the output layer neuron *Yin* is calculated as:

$$Yin[k] = Yin[k] + Z[j] \times w[j,k], \text{ for } 1<=j<=h,1<=k<=m \quad (41)$$

Step 8: The output from the output layer neuron is calculated as : $Y[k] = \text{binsig}(Yin[k])$, for $1<=k<=m$ (42)

Step 9: This output is stored in *ty*.

$$ty[I,k] = Y[k], \text{ for } 1<=k<=m \quad (43)$$

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

Step 10: $I=I+1$

If $I<mc$ goto step 2 else goto step 11.

Step 11: Now, open the *tyfile* in "read" mode.

Read tyfile into *ty* matrix.

Step 12: Now *ty* is reshaped into a row vector of dimension $(1\times (mc\times nc))$.

$$ty[(i-1) \times 4+j]=t[i,j] \text{ for } 1<=i<=mc,1<=j<=nc \quad (44)$$

This provides a row vector $ty=[ty1,ty2,\ldots.ty_{mc*nc}]$

Step 13: Now display the image represented by *ty*

4. EXPERIMENTS CONDUCTED WITH AND THE RESULTS:

4.1 Variation of PSNR with threshold value

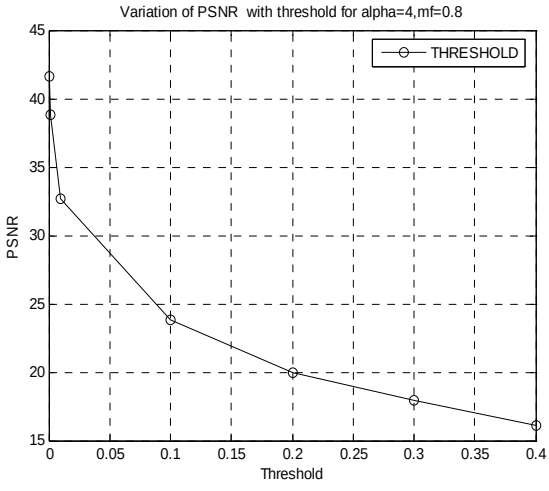
In the first experiment , the variation of PSNR values with respect to change in threshold value is clearly visible. The threshold is varied from 0.4 to 0.0001 as shown in table – I . With the reduction in the threshold value, the PSNR goes on increasing. There is also an increment seen in training time 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 16.11 to 41.64. The best PSNR value is obtained at threshold value of 0.0001 with a training time of 2567.98 seconds. 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. The chart 1 shows the variation of PSNR values with respect to threshold values in a graphical way.

TABLE 1. *Variation of PSNR with threshold*

(α =4 , *mf*= 0.8)

α	mf	Threshold	PSNR	Trainining time(sec)
4	0.8	0.4	16.11	321.90
4	0.8	0.3	17.98	352.48
4	0.8	0.2	19.92	415.98
4	0.8	0.1	23.78	554.45
4	0.8	0.01	32.66	698.87
4	0.8	0.001	38.83	1254.56
4	0.8	0.0001	41.64	2567.98

CHART 1. *PSNR with threshold*



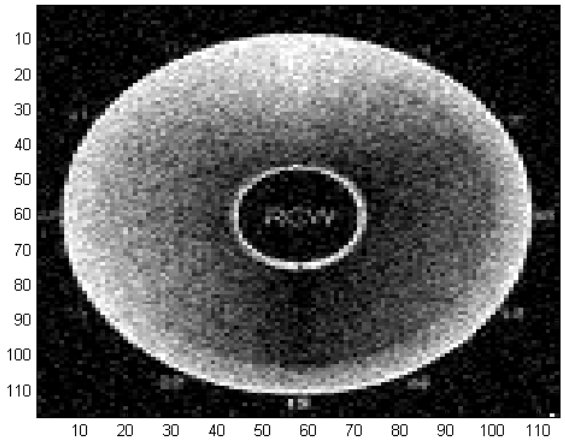


Figure 1. threshold=0.1

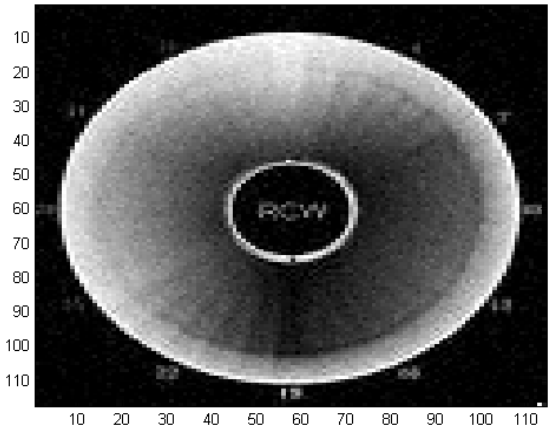


Figure 2.Threshold=0.11

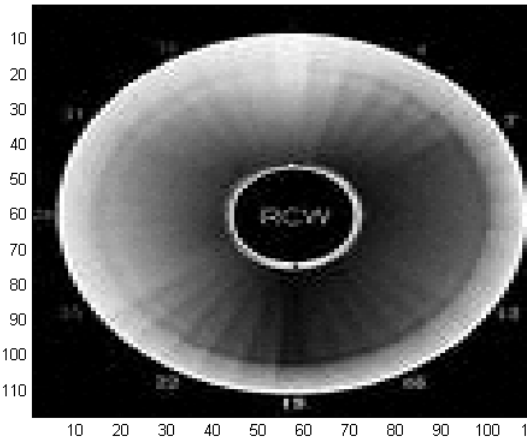


Figure 3.Threshold = 0.001

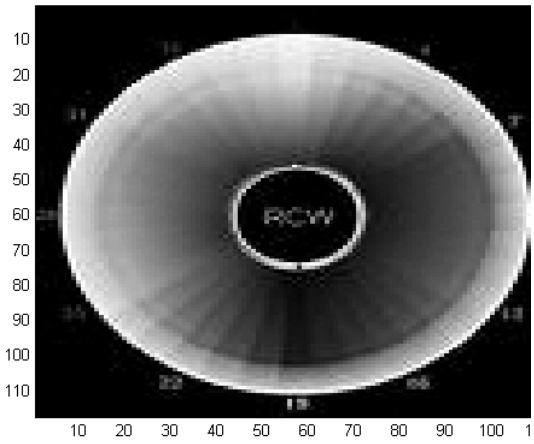


Figure 4.Threshold = 0.0001

4.2 Robustness test

In this experiment, the robustness of the watermarking scheme is shown. The cover image of Lena shown in fig. 5 contains only the random number embedded in the higher precision bits of the intensity value of the first pixel of the image. 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. The various attacks used were blurring, cropping, sharpening, rotation, 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 II 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 number was embedded in the cover image of Lena and it was also saved with the files. This number is also embedded in the cover image mainly for the purpose of authentication as discussed in later experiments.

TABLE – 2. *Robustness Test*

Attack	Threshold values			
	0.1	0.01	0.001	0.0001
	(PSNR)	(PSNR)	(PSNR)	(PSNR)
Blurred	23.78	32.66	38.83	41.64
Cropped	23.78	32.66	38.83	41.64
Sharpen	23.78	32.66	38.83	41.64
Rotation	23.78	32.66	38.83	41.64
Scaling	23.78	32.66	38.83	41.64
JPEG	23.78	32.66	38.83	41.64

4.3 Imperceptibility test

In this experiment, the test of imperceptibility is done. The cover image taken was Lena's image . The trained Neural Network weights were hidden inside this cover image only. The PSNR value of the watermarked image of Lena after the insertion of the weight values with respect to the original picture of Lena is calculated as 112.3324.

This high value of PSNR indicates, that , there is a very little deterioration in the quality of cover image by insertion of the trained network weight values. This has become possible, since the weight values were hidden in the extremely least significant bits of the fractional part of the pixel values of the cover image. Thus, the property of imperceptibility is highly preserved under this scheme.

Figure 5. Original Lena's image

Figure .6. Lena's image after weight values insertion

4.4 Authenticity test:

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.

5. CONCLUSIONS:

This paper describes the algorithm to use a Backpropagation 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 the imperceptibility of the cover image is also highly preserved. 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.

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