International Journal of Soft Computing 3 (6): 443-450, 2008
ISSN: 1816-9503

Generating Watermark Image with Authentication Using Full Cover Image and Backpropagation Neural Network

[1]Ashish Bansal and [2]Sarita Singh Bhadauria
[1]Department of IT, Mahakal Institute of Technology, Ujjain, India
[2]Department of Electronics, Madhav Institute of Technology and Science, Gwalior, India

Abstract: In this digital world, it has become very important to save a digital product from illegal copy or reproduction. The technique, which has evolved for this, is known as Digital Watermarking. Several techniques based on spatial and frequency domain have been developed. However, none of them are full proof and exhibit all the desirable properties of watermarking to a satisfactory level. There is a gradual reduction in the fidelity of the cover image with the increase in embedded information content. This study discusses a special method based on Backpropagation Neural Network, which takes a cover image at the input of a Backpropagation Neural Netowrk and the network is trained to produce the desired watermark image. The cover image is taken in the original form and is not fragmented. After training, a random number is embedded in the higher precision bit of the cover image pixel and supplied with the trained network weights for the extraction of watermark. This guarantees authentication also. During the extraction stage, the watermarked image is supplied to the input of the trained network and output watermark image is produced. As the image processing operations do not affect the weights of the Neural Network, so the watermark image is resistant to various image processing operations enhancing robustness of watermarking.

Key words: Digital watermark, Neural Net, backpropagation network

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) (Van *et al*., 1994), linear addition of watermark to cover image (Van *et al*., 1994), using mid band coefficients of DCT transformed blocks to hide watermark (Ahmidi and Safabaksh, 2004), maximizing strength of watermark using.

Discrete Wavelet Transform (DWT) techniques (Davis and Najarian, 2001), Using radial basis function (RBF) neural network to achieve maximum strength watermark (Zhang *et al*., 2003), transforming color space of cover image and embedding watermark into saturation channel (Ren *et al*., 2002), Embedding watermark in the DC components of transformed blocks (Fengsen and Bingxi, 2003) etc. Principles of neurocomputing and their usage in science and technology is well explained in Fredric and Kostanic (2001). Cox *et al*. (1996) 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*. (1994) generated a watermark using a m-sequence generator. Sehyndel *et al*. (1994) introduced a watermarking scheme using fractal codes. Bartolini *et al*. (1998) utilized the properties of human visual system and generated watermark from DCT coefficients. Kundur and Hatzinakos (1997) embedded the watermark in the wavelet domain where the strength of watermark was decided by the contrast sensitivity of the original image. Delaigle *et al*. (1998) 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 Pitas (1996) and immunity to subsampling was examined. Cox and Kilan (1996) presented a secure algorithm for watermarking images using spread-spectrum techniques. Craver and Memon (1998) 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 this study. These techniques also have the problems related to security.

The use of Neural Network for successful watermarking was effectively done in Chun (2005), where Full Counterpropagation Network (FCNN) was used to insert the watermark into synapses of FCNN rather than

Corresponding Author: Ashish Bansal, Department of IT, Mahakal Institute of Technology, Ujjain, India

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 study is an attempt to adopt backpropagation neural network for the purpose of watermarking. The inputs to be provided to the input layer are taken from a cover image. This shall help to maintain robustness, fidelity and authenticity of the watermark as depicted in later sections. Hiding a random number in the higher precision bit of the pixel value helps in the authentication of the watermarked image.

APPROACH FOR USING BACKPROPAGATION WITH COVER IMAGE AND GIVEN TARGET WATERMARK IMAGE

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

Embedding:

- The target watermark image is taken to serve as output to a Backpropagation Neural Network.
- A Backpropagation Neural Network is chosen with 1 input, 1 hidden and 1 output layer.
- 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.
- The trained weight matrix are stored in files. A random number is embedded in the higher precision bit of the cover image pixel and also in a file for the authenticity purpose.

Extraction:

- The cover image is taken and random number is extracted from the higher precision bit of the pixel intensity value.
- This random number is used to determine the authenticity of the watermarked image.
- The weights are extracted from the files and the trained neural network is reconstructed.
- The watermarked image is supplied at the input layer neurons and the final output watermark image is produced at the output layer.
- The output watermark image is correlated with the target output watermark to determined PSNR of the obtained watermark image.

Algorithm: The following conventions apply to the embedding algorithm as well as extraction algorithms given:

- M = rand (m, c) generates a random matrix M containing m rows and n columns.
- 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.
- M = binsig (M) generates a matrix containing binary sigmoid values of each value of the matrix M.
- M = binsigl (M) generates.
  binsig (M) (1-binsig (M)).
- min_threshold_error puts a lower bar on the acceptable value of error generated.

Embedding

Step 1: Let the target watermark image be given as:

$$\text{timage} = [t_{11}, t_{12}, \ldots . t_{ij}, \ldots \ldots t_{mc\times nc}] \quad \text{for } 1<=i<=mc,\ 1<=j<=nc \tag{1}$$

where,

mc = Number of rows in the target image.
nc = Number of columns in the target image.

The cover image used to produce the target watermark image be given as:

$$\text{cimage} = [c_{11}, c_{12}, \ldots . c_{ij}, \ldots \ldots c_{mc\times nc}] \quad \text{for } 1<=i<=mc,\ 1<=j<=nc \tag{2}$$

where,

mc = Number of rows in the cover image.
nc = Number of columns in the cover image.

Step 2: Now, timage is reshaped as a row vector containing mc×nc number of columns.

$$\text{timage}[(i-1)\times nc+j] = t[i, j] \text{ for } 1<=i<=mc,\ 1<=j<=nc \tag{3}$$

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

Step 3: Now, cimage is reshaped as a row vector containing mc×nc number of columns.

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

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:

$$v = \text{rand}(n, h) - 0.5 \quad (5)$$

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

$$w = \text{rand}(h, m) - 0.5 \quad (6)$$

The initial bias of hidden layer neurons is set as:

$$b1 = \text{rand}[1, h] - 0.5 \quad (7)$$

The initial bias of output layer neurons is set as:

$$b2 = \text{rand}[1, m] - 0.5 \quad (8)$$

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.

$$v1 = \text{zeros}(n, h) \quad (9)$$

$$w1 = \text{zeros}(h, m) \quad (10)$$

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.

$$con = 1 \quad (11)$$

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

$$epoch = 0 \quad (12)$$

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

Step 5: Repeat the steps from 6-12 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.

$$e = 0 \quad (13)$$

Step 7: Now to pick up each row of cimage for training, repeat the steps from 8-10 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.

$$Zin(j) = b1(j) \text{ for } 1 <= j <= h \quad (14)$$

The net input Zin is calculated as:

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

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

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

Let, Yin represents the net input to the output layer. Yin is initialized with a bias b2.

$$Yin(k) = b2(k) \text{ for } 1 <= k <= m \quad (17)$$

The net input Yin is calculated as:

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

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

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

This output is stored in a matrix ty.

$$ty(I, k) = Y(k) \text{ for } 1 <= k <= m \quad (20)$$

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

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

where,

timage (I, k)-Y (k) = The error at the kth neuron in the output layer.

The weights at the output layer are adjusted by:

$$delw(j, k) = alpha \times delk(k \times z(j) + mf \times (w(j, k) - w1(j, k)) \quad \text{for } 1 <= k <= m, 1 <= j <= h \quad (21)$$

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

$$delb2(k) = alpha \times delk(k), \text{ for } 1 <= k <= m \quad (22)$$

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

$$delinj(j) = 0 \text{ for } 1 <= j <= h \quad (23)$$

delinj is modified with the help of delk.

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

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

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

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

The modifications in the weight matrix v is given by:

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

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

$$delb1[j] = alpha \times delj[j], \text{ for } 1 <= j <= h \quad (27)$$

Now, initial weights w and v are stored in w1 and v1, 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.

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

The weight matrix v is updated.

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

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.

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

Step 10:

$$I = I + 1, \text{ goto step 7 if } I < mc \quad (34)$$

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

$$\text{If } e < min\_threshold\_error, con = 0 \quad (35)$$

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 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 b1 is stored in b1file.
- The bias matrix b2 is stored in b2file

Now, all the files are closed.

A random number is stored in the higher precision bit of the pixel value of the cover image.

Now, this watermarked image is supplied with the trained weight matrix files for the watermark extraction algorithm.

Extraction

Step 1: The hidden random number is extracted from the higher precision bit of the watermarked image pixel and also from the supplied file and a match is seen. If the 2 are same then the authenticity of the watermark is preserved.

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: 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.
- b1 is populated from b1file.
- b2 is populated from b2file.

Step 4: For each value of I from 1 to mc, perform the steps from 5-12.

Step 5: Initialize Zin with the bias b1.

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

Step 6: 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 7: The output of the hidden layer neuron is calculated as:

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

Step 8: Initialise Yin with the bias b2.

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

Step 9: 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 10: The output from the output layer neuron is calculated as:

$$Y[k] = binsig(Yin[k]), \text{ for } 1 <= k <= m \quad (42)$$

Step 11: 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 12: I = I+1
If I<mc goto step 4 else goto step 13.

Step 13:
Close all files.
Now, open the tyfile in read mode.
Read tyfile into ty matrix.

Step 14: Now ty is reshaped into a row vector of dimension (1 × (mc×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, ….ty_{mc*nc}]

Step 15: Now display the image represented by ty

EXPERIMENTS CONDUCTED WITH AND THE RESULTS

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- 0.0001 as shown in Table 1. 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 16.11-41.64. The best PSNR value is obtained at threshold value of 0.0001 with a training time of 2567.98 sec. Figure 1-4, show the extracted watermark image corresponding to threshold values of 0.1, 0.01, 0.001 and 0.0001, respectively. The Fig. 5 shows the variation of PSNR values with respect to threshold values in a graphical way.

In this experiment, the robustness of the watermarking scheme is shown. The cover image of Lena shown in Fig. 6 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

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

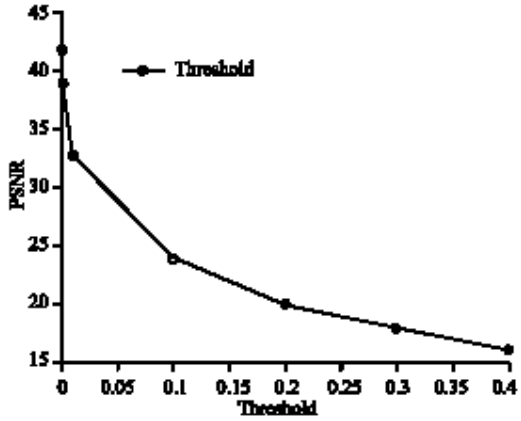

Fig. 1: Variation of PSNR with thershold for $\alpha = 4$, mf = 0.8

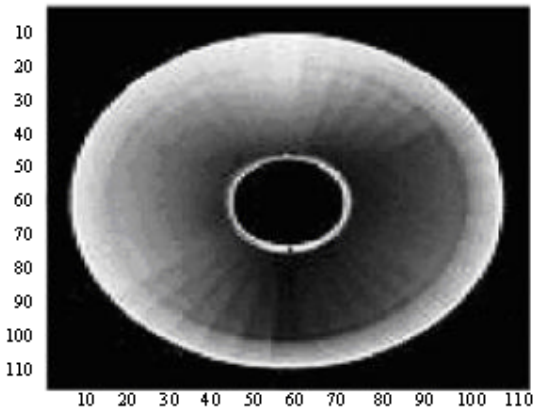

Fig. 4: Threshold = 0.001

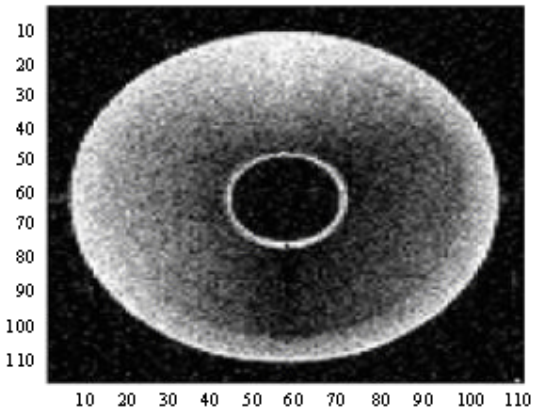

Fig. 2: Threshold = 0.1

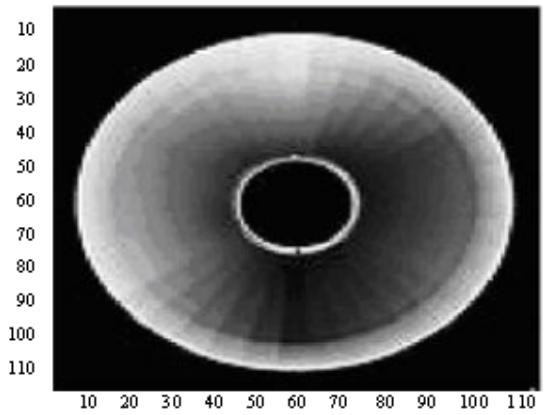

Fig. 5: Threshod = 0.0001

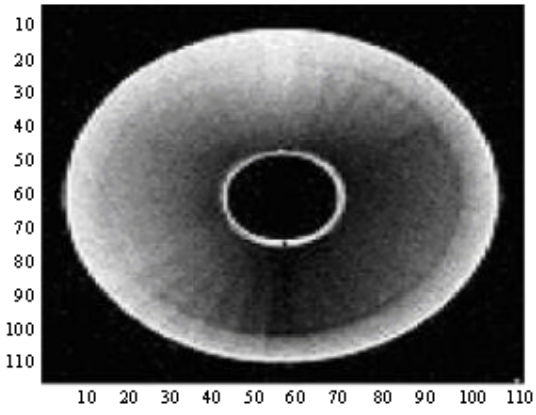

Fig. 3: Threshold = 0.01

Fig. 6: Original Lena's image

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 0.1-0.0001, respectively. The Table 2 shows the obtained values of PSNR for each of these attacks. It

is seen that these values are exactly same as shown in Table 1. 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.

Robustness test: In this experiment, the test of imperceptibility is done.

The cover image taken was Lena's image. A random number was hidden in the higher precision of first pixel value. Figure 6 shows the original Lena's image and Fig. 7 shows the same image with embedded random number.

Let the cover image (Lena's image) be given as:

$$Y = [y_{11}, y_{12}, \ldots . y_{ij}, \ldots \ldots y_{mc-nc}]$$

The first pixel value Y (1, 1) = 136 is changed to

$$Y (1, 1) = 136.0010.$$

The last 2 bits represent the hidden random number.

Table 2: PSNR values of the watermark image

	Threshold values			
Attack	0.1 (PSNR)	0.01 (PSNR)	0.001 (PSNR)	0.0001 (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

Fig. 7: Lena's image with embedded random key

This value is extracted and used for testing authenticity of the watermarked image.

The PSNR value of the watermarked image of Lena after the insertion of the random number, with respect to the original picture of Lena is calculated as 148.4380.

This high value of PSNR indicates, that, there is a very little deterioration in the quality of cover image by insertion of the random number. Thus, the property of imperceptibility is highly preserved under this scheme.

Authenticity test: The hidden random number 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 number stored in the file during the algorithm. If the 2 values match, authenticity is preserved, otherwise, the authenticity is suspected. Thus, authenticity feature is also well preserved in this scheme.

CONCLUSION

In this study, Backpropagation Neural Network has been used to map a cover image into corresponding output target watermark image. A random number embedded into the cover image has helped to preserve the authenticity requirements of the watermarking scheme. As seen from the results, robustness is also preserved, as the image processing operations do not affect the trained neural network weights in this scheme. However, there is a limitation due to a very large training time required in this scheme. Thus, Backpropagation Neural Network may also be employed to provide a successful watermarking scheme.

REFERENCES

Ahmidi, N. and R. Safabaksh, 2004. A Novel DCT Based Approach for Secure Color Image Watermarking. In: Proc. ITCC 2004 Int. Conf. Inform. Technol. Coding and Comput., 2: 709-713.

Bartolini, F., M. Barni, V. Cappellini and A. Piva, 1998. Mask Building for Perceptually Hiding Frequency Embedded Watermarks. In: Proc. Int. Conf. Image Proc., I: 450-454.

Chun-Yu-Chang, 2005. The Application of a Full Counterpropagation Neural Network to Image Watermarking, IEEE.

Cox, I. and J. Kilan, 1996. Secure Spread Spectrum Watermarking for Images, Audio and Video. In: Proc. IEEE Int. Conf., Image Proc., 3: 243-246.

Cox, J. and J. Kilian, 1996. A Secure Robust Watermark for Multimedia. In Proc. 1st International Workshop. Lecture Notes in Comput. Sci., 1174: 185-206.

Craver, S. and N. Memon, 1998. Resolving Rightful Ownership with Invisible Watermarking Techniques: Limitations, Attacks and Implications. IEEE Trans., 16 (4): 573-586.

Davis, K.J.and K. Najarian, 2001. Maximizing Strength of Digital Watermarks Using Neural Networks. In: Proc. Int. Joint Conf. Neural Network, 4: 2893-2898.

Delaigle, J., C. De Vleeschouwer and B. Macq, 1998. Psychovisual Approach to Digital Picture Watermarking. J. Elec. Imaging, 7 (3): 628-640.

Fengsen Deng and Bingxi Wang, 2003. A Novel Technique for Robust Image Watermarking in the DCT Domain. In: Proc. 2003 Int. Conf. Neural Networks and Signal Proc., 2: 1525-1528.

Fredric, M.H. and I. Kostanic, 2001. Principles of Neurocomputing for Science and Engineering. Mc.GrawHill, Singapore, pp: 136-140.

Kundur, D. and D. Hatzinakos, 1997. A Robust Digital Image Watermarking Method using Wavelet-Based Fusion. In: Proc. IEEE Int. Conf. Image Proc., 1: 544-547.

Pitas, I., 1996. A Method for Signature Casting on Digital Images. In: Proc. IEEE Int. Conf. Image Proc., 3: 215-218.

Ren-Junn Hwand, Chuan-Ho Kao and Rong-Chi Chang, 2002. Watermark in Color Image. In: Proc. 1st Int. Symp. Cyber Worlds, pp: 225-229.

Schyndel, R., A. Tirkel and C. Osborne, 1994. A Digital Watermark. In: Proc. IEEE Int. Conf. Image Proc., 2: 86-90.

Van Schyndel, R.G., A.Z. Tirkel and C.F. Osborene, 1994. A Digital Watermark. In: Proc. IEEE Int. Conf. Image Proc., 2: 86-92.

Zhang Zhi Ming, Li Rong-Yan and Wang Lei, 2003. Adaptive Watermark Scheme with RBF Neural Networks. In: Proc. 2003 Int. Conf. Neural Networks and Signal Process., 2: 1517-1520.