A New Hybrid System for Information Security

Hazem M. El-Bakry
Faculty of Computer Science & Information Systems,
Mansoura University, EGYPT
helbakry20@yahoo.com

Nikos Mastorakis
Technical University of Sofia,
BULGARIA

Abstract – For the purpose of efficient observation and monitoring in information security, the input image is required to be transferred from one place to another for decision making (higher administration). In this paper, the hybrid kRPCA method is used for efficient compression. However, some pixels in the input image may be missed or distorted during the transmission process. Therefore, Hopfield neural networks are used for retrieving the original of distorted images. Then fast feedforward neural networks (FFNNs) are applied for face detection in the received image. The speed of these neural networks is accelerated by modifying its algorithm. This is done by applying cross correlation in the frequency domain between the received image and the weights of neural networks. The result of cross correlation implemented in the frequency domain is the same as that one obtained in time domain. Moreover, the speed of operation is faster than performing cross correlation in time domain. It is proved mathematically and practically that the proposed algorithm is fast and efficient in retrieving missed pixels in distorted images.

Keywords: Hopfield Neural Networks, FFNNs, Cross Correlation, Frequency Domain, k-PCA

I. Introduction

Retention of noised or distorted images is very important for many different applications such as information security. For example, the increase in social web applications and the semantic web have inspired the development of several web-based image annotation tools. In Biometrics, the images of faces, fingerprints, or hand geometry may be noised or distored especially in developed countries with large number of hardworkers and farmers. Among different biometrics techniques, face recognition is the only suitable method for police covert surveillance [4,7,24,26,28]. The human face is a complex pattern. Finding human faces automatically in a scene is a difficult yet significant problem. It is the first step in fully automatic human face recognition system. Face detection is the fundamental step before the face recognition or identification procedure. Its reliability and time response have a major influence on the performance and usability of the whole face recognition system. For web indexation applications, the processing time must be kept as low as possible as the number of images on the web increases continuously [59-61].

After the image which may contain the human face is captured, it is required to be transferred through computer networks to higher administration for monitoring. Therefore, an efficient compression algorithm is needed. In recent years, principal component analysis (PCA) has attracted great attention in image compression. However, since the compressed image data include both the transformation matrix (the eigenvectors) and the transformed coefficients, PCA cannot produce the performance like DCT (discrete cosine transform) in respect of compression ratio. In using DCT, we need only to preserve the coefficients after transformation, because the transformation matrix is universal in the sense that it can be used to compress all images. In [38-41] a universal PCA by proposing a hybrid method called k-PCA was presented. The basic idea is to construct k sets of eigenvectors for different image blocks with distinct characteristics using some training data. The k sets of eigenvectors are then used to compress all images. Vector quantization (VQ) is adopted to split the training data space.

After the received image is decompressed, it may be noised or distorted for many reasons. Therefore, it is required to retrieve the original image. The efficiency of the image retrieving system has a great influence on the next step which is face recognition systems. It is known that Hopfield neural networks can retrieve patterns even with noise, distotion or deformation in shape. So, in this research, Hopfield neural networks are used to retrieve noised, distorted or missed pixles to get the original image.

Among other techniques of face detection, neural networks are efficient face detectors [59-61]. One of the main objectives of this research is to reduce the response time of neural networks for face detection. The purpose
is to perform the testing process in the frequency domain instead of the time domain. Our approach was successfully applied for fast detection of many different patterns [1-28].

This paper is organized as follows: the k-PCA approach by combining VQ and PCA is discussed in section 2. Retrieval of missed pixels by using Hopfield neural networks is introduced in section 3. Fast face detection by using FFNs is presented in section 4. Finally, conclusion is given in section 5.

2. Image Compression by using K-PCA

By implementing PCA it is known that, we should build one particular transformation matrix consisting of eigenvectors. So, when reconstructing the image, not only the transformed coefficients but also the transform matrix is required. Furthermore PCA is a linear approach; it cannot approximate all areas of the image equally well. In other words, one PCA cannot simultaneously capture the features of all regions. To resolve the above problems, MPC has been studied [31-37]. The procedure is as follows: before PCA, divide the problem space into a number of subspaces, and then find a set of eigenvectors for each sub-space. If enough training data are given, MPC can construct a system which maintains a good generality. It is interesting to note that an MPC can be used as a universal encoder if the generalization ability is high enough. In this case, we do not have to preserve the MPC parameters in the compressed data. Only the transformed coefficients (the output of the system for each input image block) are needed.

So far researches have been focused on how to divide the problem space efficiently. In [37,38], Donny proposed an optimally adaptive transform coding method. It is composed of a number of GHA neural networks. Fig. 1 illustrates how the appropriate GHA is selected to learn from the current input vector. The training algorithm is as follows [37,38]:

- **Step 1:** Initialize (at random) K transformation matrices \(W_1, W_2, \ldots, W_K\), where \(W_j\) is the weight matrix of the j-th GHA network.

- **Step 2:** For each training input vector \(x\), classify it to the i-th sub-space, if

\[
P_i x = \max_{j=1}^{K} P_j x
\]

where \(P_i = W_i^T W_i\).

Update the weights according to the following rule:

\[
W_i^{\text{new}} = W_i^{\text{old}} + \alpha Z (x, W_i^{\text{old}})
\]

Where \(\alpha\) is the learning rate and \(Z\) is a GHA learning rule which converges to the principal components.

- **Step 3:** Iteratively implement the above training procedure until the weights are stable.

In [37], the training parameters are: 1) the number of sub-spaces is 64 and 2) the number of training iterations is 80,000. Note that to use the MPC as a universal encoder; we must train it using many data. The above algorithm clearly is not good enough because it is too time consuming. In [38], several methods were proposed to speed up the training process and decrease the distortion. These methods include growth by class insertion, growth by components addition and tree structured network. The essential issue is that the convergent speed of GHA is very slow [38-41].

To enhance the operation of PCA, the authors [38-41] proposed K-PCA. The encoding and decoding procedure of the K-PCA method is given in Fig.2.

- **Step 1:** Divide the input image into \(n \times n\) small blocks \((n=8)\). For the entire input data, find an 8-D PCA encoder. By so doing, the dimension of the problem space can be reduced from 64 to 8.

- **Step 2:** Find a codebook with \(k\) \((k=64\) in our experiments\) code words using the LBG algorithm, for the 8-D vectors obtained in the last step, and record the index of each input vector.

- **Step 3:** Based on the codebook, the problem space can be divided into \(k\) clusters. For each cluster, we can find an M-D \((M=4)\) PCA encoder.

- **Step 4:** For each input vector, compress it to an 8-D vector using the PCA encoder found in Step 1, then find the index of the nearest code word found in Step 2, and finally compress it to an M-D vector. The M-D vector along with the index of the nearest code word is used as the code of the input vector.

The purpose of Step 1 is to reduce the computational cost of VQ. Through experiments it has been found that an 8-D PCA encoder can represent the original image
very well. The codebook obtained based on the 8-D vectors performs almost the same as that obtained from the original 64-D vectors. The above encoding method is called k-PCA. Note that if k-PCA is trained by using enough data, it can be used as a universal encoder, and do not have to include the eigenvectors into the compressed data. Thus, the compression ratio can be increased.

The reconstruction (decoding) procedure is as follows [38-41]:

- **Step 1:** Read in the codes one by one.
- **Step 2:** Find the basis vectors for the cluster specified by the index, and transform the M-D vector back to the 8-D vector.
- **Step 3:** Transform the 8-D vector back to \( n \times n \)-D vector, and put it to the image in order.

### 3. Retrieval of Distorted Images by using Hopfield Neural Networks

An image retrieval system is a computer system for browsing, searching and retrieving images from a large database of digital images. Most traditional and common methods of image retrieval utilize some method of adding metadata such as captioning, keywords, or descriptions to the images so that retrieval can be performed over the annotation words. Manual image annotation is time-consuming, laborious and expensive; to address this, there has been a large amount of research done on automatic image annotation. Here, hopfield neural networks are used to retrieve distorted images. It is known that hopfield neural networks can retrieve patterns even with noise, distortion or deformation in shape.

One of the most important functions of our brain is the laying down and recall of memories. It is difficult to imagine how we could function without both short and long term memory. The absence of short term memory would render most tasks extremely difficult if not impossible - life would be punctuated by a series of one time images with no logical connection between them. Equally, the absence of any means of long term memory would ensure that we could not learn by past experience. Indeed, much of our impression of self depends on remembering our past history [42-45].

Our memories function in what is called an associative or content-addressable fashion. That is, a memory does not exist in some isolated fashion, located in a particular set of neurons. All memories are in some sense strings of memories - you remember someone in a variety of ways - by the color of their hair or eyes, the shape of their nose, their height, the sound of their voice, or perhaps by the smell of a favourite perfume. Thus memories are stored in association with one another. These different sensory units lie in completely separate parts of the brain, so it is clear that the memory of the person must be distributed throughout the brain in some fashion. Indeed, PET scans reveal that during memory recall there is a pattern of brain activity in many widely different parts of the brain [42-48].

Notice also that it is possible to access the full memory (all aspects of the person's description for example) by initially remembering just one or two of these characteristic features. We access the memory by its contents not by where it is stored in the neural pathways of the brain. This is very powerful; given even a poor photograph of that person we are quite good at reconstructing the persons face quite accurately. This is very different from a traditional computer where specific facts are located in specific places in computer memory.

If only partial information is available about this location, the fact or memory cannot be recalled at all [42-48].

Theoretical physicists are an unusual lot, acting like gunslingers in the old West, anxious to prove themselves against a really good problem. And there aren’t that many really good problems that might be solvable. As soon as Hopfield pointed out the connection between a new and important problem (network models of brain function) and an old and well-studied problem (the Ising model), many physicists rode into town, so to speak, with the intention of shooting the problem full of holes and then, the brain understood, riding off into the sunset looking for a newer, tougher problem. (Who was that masked physicist?).

Hopfield made the portentous comment: ‘This case is isomorphic with an Ising model,’ thereby allowing a deluge of physical theory (and physicists) to enter neural network modeling. This flood of new participants transformed the field. In 1974 Little and Shaw made a similar identification of neural network dynamics with the Ising model, but for whatever reason, their idea was not widely picked up at the time. Unfortunately, the problem of brain function turned out to be more difficult than expected, and it is still unsolved, although a number of interesting results about Hopfield nets were proved. At present, many of the traveling theoreticians have traveled on [49].

The Hopfield neural network is a simple artificial network which is able to store certain memories or patterns in a manner rather similar to the brain - the full pattern can be recovered if the network is presented with only partial information. Furthermore there is a degree
of stability in the system - if just a few of the connections between nodes (neurons) are severed, the recalled memory is not too badly corrupted - the network can respond with a "best guess". Of course, a similar phenomenon is observed with the brain - during an average lifetime many neurons will die but we do not suffer a catastrophic loss of individual memories - our brains are quite robust in this respect (by the time we die we may have lost 20 percent of our original neurons) [42-58].

The nodes in the network are vast simplifications of real neurons - they can only exist in one of two possible "states" - firing or not firing. Every node is connected to every other node with some strength. At any instant of time a node will change its state (i.e start or stop firing) depending on the inputs it receives from the other nodes [42-58].

If we start the system off with a any general pattern of firing and non-firing nodes then this pattern will in general change with time. To see this think of starting the network with just one firing node. This will send a signal to all the other nodes via its connections so that a short time later some of these other nodes will fire. These new firing nodes will then excite others after a further short time interval and a whole cascade of different firing patterns will occur. One might imagine that the firing pattern of the network would change in a complicated perhaps random way with time. The crucial property of the Hopfield network which renders it useful for simulating memory recall is the following: we are guaranteed that the pattern will settle down after a long enough time to some fixed pattern. Certain nodes will be always "on" and others "off". Furthermore, it is possible to arrange that these stable firing patterns of the network correspond to the desired memories we wish to store! [42-58].

The reason for this is somewhat technical but we can proceed by analogy. Imagine a ball rolling on some bumpy surface. We imagine the position of the ball at any instant to represent the activity of the nodes in the network. Memories will be represented by special patterns of node activity corresponding to wells in the surface. Thus, if the ball is let go, it will execute some complicated motion but we are certain that eventually it will end up in one of the wells of the surface. We can think of the height of the surface as representing the energy of the ball. We know that the ball will seek to minimize its energy by seeking out the lowest spots on the surface -- the wells. Furthermore, the well it ends up in will usually be the one it started off closest to. In the language of memory recall, if we start the network off with a pattern of firing which approximates one of the "stable firing patterns" (memories) it will "under its own steam" end up in the nearby well in the energy surface thereby recalling the original perfect memory. The smart thing about the Hopfield network is that there exists a rather simple way of setting up the connections between nodes in such a way that any desired set of patterns can be made "stable firing patterns". Thus any set of memories can be burned into the network at the beginning. Then if we kick the network off with any old set of node activity we are guaranteed that a "memory" will be recalled. Not too surprisingly, the memory that is recalled is the one which is "closest" to the starting pattern. In other words, we can give the network a corrupted image or memory and the network will "all by itself" try to reconstruct the perfect image. Of course, if the input image is sufficiently poor, it may recall the incorrect memory - the network can become "confused" - just like the human brain. We know that when we try to remember someone's telephone number we will sometimes produce the wrong one! Notice also that the network is reasonably robust - if we change a few connection strengths just a little the recalled images are "roughly right". We don't lose any of the images completely [42-58].

As with the Linear Associative Memory, the "stored patterns" are represented by the weights. To be effective, the patterns should be reasonably orthogonal. The basic Hopfield model can be described as follows [49]:

- N neurons, fully connected in a cyclic fashion:
- Values are +1, -1.
- Each neuron has a weighted input from all other neurons.
- The weight matrix $w$ is symmetric ($w_{ij} = w_{ji}$) and diagonal terms (self-weights $w_{ii} = 0$).
- Activation function on each neuron $i$ is:
  \[ f(\text{net}) = \text{sgn}(\text{net}) = \begin{cases} 1 & \text{if } \text{net} > 0 \\ -1 & \text{if } \text{net} < 0 \end{cases} \]

where:
  \[ \text{net}_i = \sum w_{ij} x_j \] (4)

- If $\text{net} = 0$, then the output is the same as before, by convention.
- There are no separate thresholds or biases. However, these could be represented by units that have all weights $= 0$ and thus never change their output.
- The energy function is defined as:
  \[ E(y_1, y_2, \ldots, y_n) = -\sum \sum w_{ij} y_i y_j \] (5)

where $(y_1, y_2, \ldots, y_n)$ is outputs, $w_{ij}$ is the weight neuron $i$, and the double sum is over $i$ and $j$. 
Different imagess are stored in Hopfield neural network. In the testing process, the missed/distorted pixels (if any) are retrieved.

4. Fast Image Face Detection by using Feedforward Neural Networks

First neural networks are trained to classify sub-images which contain face from those which do not and this is done in the spatial domain. In the test phase, each sub-image in the input image (under test) is tested for the presence or absence of human faces. At each pixel position in the input image each sub-image is multiplied by a window of weights, which has the same size as the sub-image. This multiplication is done in the spatial domain. The outputs of neurons in the hidden layer are multiplied by the weights of the output layer. When the final output is high this means that the sub-image under test contain human motion and vice versa. Thus, we may conclude that this searching problem is cross correlation in the spatial domain between the image under test and the input weights of neural networks.

In this section, a fast algorithm for detecting human motion based on two dimensional cross correlations that take place between the tested image and the sliding window (20x20 pixels) is described. Such window is represented by the neural network weights situated between the input unit and the hidden layer. The convolution theorem in mathematical analysis says that a convolution of $f$ with $h$ is identical to the result of the following steps: let $F$ and $H$ be the results of the Fourier transformation of $f$ and $h$ in the frequency domain. Multiply $F$ and $H$ in the frequency domain point by point and then transform this product into spatial domain via the inverse Fourier transform [30]. As a result, these cross correlations can be represented by a product in the frequency domain. Thus, by using cross correlation in the frequency domain a speed up in an order of magnitude can be achieved during the detection process [1-28].

In the detection phase, a sub-image $X$ of size $m \times z$ (sliding window) is extracted from the tested image, which has a size $P \times T$, and fed to the neural network. Let $W_P$ be the vector of weights between the input sub-image and the hidden layer. This vector has a size of $m \times z$ and can be represented as $m \times z$ matrix. The output of hidden neurons $h_i$ can be calculated as follows:

$$ h_i = g\left(\sum_{j=1}^{m} \sum_{k=1}^{z} W_{ij}(j,k)X(j,k) + b_i\right) $$

where $g$ is the activation function and $h_i$ is the bias of each hidden neuron $(i)$. Eq.1 represents the output of each hidden neuron for a particular sub-image $I$. It can be computed for the whole image $\Psi$ as follows:

$$ h_i(\Psi) = g\left(\sum_{j=-m/2}^{m/2} \sum_{k=-z/2}^{z/2} W_{ij}(j,k)\Psi(u+j,v+k)+b_i\right) $$

Eq.7 represents a cross correlation operation. Given any two functions $f$ and $g$, their cross correlation can be obtained by [30]:

$$ g(x,y) \otimes f(x,y) = \left(\sum_{m=-\infty}^{\infty} \sum_{z=-\infty}^{\infty} g(m,z)f(x+m,y+z)\right) $$

Therefore, Eq.8 can be written as follows [1-28]:

$$ h_i = g(W_i \otimes \Psi + b_i) $$

where $h_i$ is the output of the hidden neuron $(i)$ and $h_i(\Psi)$ is the activity of the hidden unit $(i)$ when the sliding window is located at position $(u,v)$ in the input image $\Psi$ and $(u,v) \in [P-m+1,T-n+1]$.

Now, the above cross correlation can be expressed in terms of the Fourier Transform [1-28]

$$ W_i \otimes \Psi = F^{-1}\{F[\Psi] \cdot F^*[W_i]\} $$

(*) means the conjugate of the FFT for the weight matrix. Hence, by evaluating this cross correlation, a speed up ratio can be obtained comparable to conventional neural networks. Also, the final output of the neural network can be evaluated as follows [1-28]:

$$ O(u,v) = g\left(\sum_{i=1}^{q} W_{o} (i)h_i(u,v)+b_o\right) $$

where $q$ is the number of neurons in the hidden layer. $O(u,v)$ is the output of the neural network when the sliding window located at the position $(u,v)$ in the input image $\Psi$. $W_o$ is the weight matrix between hidden and output layer.

The complexity of cross correlation in the frequency domain can be analyzed as follows:
1. For a tested image of \(NxN\) pixels, the 2D-FFT requires a number equal to \(N^2 \log_2 N^2\) of complex computation steps. Also, the same number of complex computation steps is required for computing the 2D-FFT of the weight matrix for each neuron in the hidden layer.

2. At each neuron in the hidden layer, the inverse 2D-FFT is computed. So, \(q\) backward and \((1+q)\) forward transforms have to be computed. Therefore, for an image under test, the total number of the 2D-FFT to compute is \((2q+1)N^2 \log_2 N^2\).

3. The input image and the weights should be multiplied in the frequency domain. Therefore, a number of complex computation steps equal to \(qN^2\) should be added.

4. The number of computation steps required by the fast neural networks is complex and must be converted into a real version. It is known that the two dimensional Fast Fourier Transform requires \((N^2/2) \log_2 N^2\) complex multiplications and \(N^2 \log_2 N^2\) complex additions [29]. Every complex multiplication is realized by six real floating point operations and every complex addition is implemented by two real floating point operations. So, the total number of computation steps required to obtain the 2D-FFT of an \(NxN\) image is:

\[
\rho = 6(N^2/2) \log_2 N^2 + 2(N^2 \log_2 N^2) \tag{12}
\]

which may be simplified to:

\[
\rho = 5N^2 \log_2 N^2 \tag{13}
\]

Performing complex dot product in the frequency domain also requires \(6qN^2\) real operations.

5. In order to perform cross correlation in the frequency domain, the weight matrix must have the same size as the input image. Assume that the input object has a size of \((nxn)\) dimensions. So, the search process will be done over sub-images of \((nxn)\) dimensions and the weight matrix will have the same size. Therefore, a number of zeros \(= (N^2 - n^2)\) must be added to the weight matrix. This requires a total real number of computation steps \(= q(N^2 - n^2)\) for all neurons. Moreover, after computing the 2D-FFT for the weight matrix, the conjugate of this matrix must be obtained. So, a real number of computation steps \(= qN^2\) should be added in order to obtain the conjugate of the weight matrix for all neurons. Also, a number of real computation steps equal to \(N\) is required to create butterflies complex numbers \((e^{-i2\pi kN/N})\), where \(0 < K < L\). These \((N/2)\) complex numbers are multiplied by the elements of the input image or by previous complex numbers during the computation of the 2D-FFT. To create a complex number requires two real floating point operations. So, the total number of computation steps required for the fast neural networks becomes:

\[
\sigma = (2q+1)(5N^2 \log_2 N^2) + 6qN^2 + q(N^2 - n^2) + qN^2 + N \tag{14}
\]

which can be reformulated as:

\[
\sigma = (2q+1)(5N^2 \log_2 N^2) + q(8N^2 - n^2) + N \tag{15}
\]

6. Using a sliding window of size \(nxn\) for the same image of \(NxN\) pixels, \(q(2n^2 - 1)(N - n + 1)^2\) computation steps are required when using traditional neural networks for object detection process. The theoretical speed up factor \(\eta\) can be evaluated as follows:

\[
\eta = \frac{q(2n^2 - 1)(N - n + 1)^2}{(2q+1)(5N^2 \log_2 N^2) + q(8N^2 - n^2) + N} \tag{16}
\]

The theoretical speed up ratio Eq. 16 with different sizes of the input image and different size weight matrices is listed in Table 1. Practical speed up ratio for manipulating images of different sizes and different in size weight matrices is listed in Table 2 using 2.7 GHz processor and MATLAB. An interesting property with FFNNs is that the number of computation steps does not depend on either the size of the input sub-image or the size of the weight matrix \((n)\). The effect of \((n)\) on the number of computation steps is very small and can be ignored. This is in contrast to CNNs in which the number of computation steps is increased with the size of both the input sub-image and the weight matrix \((n)\).

5. Conclusion

A high quality image transmission system has been presented. The proposed system has been applied in information security. For covert surveillance, it is required to transfer the input image from one place to another. To achieve efficient image compression a hybrid approach called k-PCA has been used. It is well trained universal eigenvectors act as a common transformation matrix like cosine function in DCT, and the VQ has been used to divide the training data into k clusters. A pre-PCA has also been used to reduce the time for building the VQ codebook. Missed, noisy and distorted pixels have been retrieved perfectly by using Hopfield neural networks. After that a fast algorithm for face detection in the received image has been introduced. Such algorithm has used the designed FFNNs. The operation of these networks depends on performing cross correlation in the frequency domain between input image and the input weights of FFNNs. It has been proved mathematically and practically that the number
of computation steps required for the presented FFNNs is less than that needed by CNNs. Simulation results using MATLAB has confirmed the theoretical computations.

References


[21] Hazem M. El-Bakry, and Qiangfu Zhao, "Fast Normalized Neural Processors For Pattern Detection Based on Cross Correlation Implemented in the Frequency Domain," Journal of Research and


Fig. 1: Basic structure of the MPC.

Fig. 2: The flow-chat of the K-PCA [9-12].
Table 1: The theoretical speed up ratio for images with different sizes.

<table>
<thead>
<tr>
<th>Image size</th>
<th>Speed up ratio (n=20)</th>
<th>Speed up ratio (n=25)</th>
<th>Speed up ratio (n=30)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100x100</td>
<td>3.67</td>
<td>5.04</td>
<td>6.34</td>
</tr>
<tr>
<td>200x200</td>
<td>4.01</td>
<td>5.92</td>
<td>8.05</td>
</tr>
<tr>
<td>300x300</td>
<td>4.00</td>
<td>6.03</td>
<td>8.37</td>
</tr>
<tr>
<td>400x400</td>
<td>3.95</td>
<td>6.01</td>
<td>8.42</td>
</tr>
<tr>
<td>500x500</td>
<td>3.89</td>
<td>5.95</td>
<td>8.39</td>
</tr>
<tr>
<td>600x600</td>
<td>3.83</td>
<td>5.88</td>
<td>8.33</td>
</tr>
<tr>
<td>700x700</td>
<td>3.78</td>
<td>5.82</td>
<td>8.26</td>
</tr>
<tr>
<td>800x800</td>
<td>3.73</td>
<td>5.76</td>
<td>8.19</td>
</tr>
<tr>
<td>900x900</td>
<td>3.69</td>
<td>5.70</td>
<td>8.12</td>
</tr>
<tr>
<td>1000x1000</td>
<td>3.65</td>
<td>5.65</td>
<td>8.05</td>
</tr>
<tr>
<td>1100x1100</td>
<td>3.62</td>
<td>5.60</td>
<td>7.99</td>
</tr>
<tr>
<td>1200x1200</td>
<td>3.58</td>
<td>5.55</td>
<td>7.93</td>
</tr>
<tr>
<td>1300x1300</td>
<td>3.55</td>
<td>5.51</td>
<td>7.93</td>
</tr>
<tr>
<td>1400x1400</td>
<td>3.53</td>
<td>5.47</td>
<td>7.82</td>
</tr>
<tr>
<td>1500x1500</td>
<td>3.50</td>
<td>5.43</td>
<td>7.77</td>
</tr>
<tr>
<td>1600x1600</td>
<td>3.48</td>
<td>5.43</td>
<td>7.72</td>
</tr>
<tr>
<td>1700x1700</td>
<td>3.45</td>
<td>5.37</td>
<td>7.68</td>
</tr>
<tr>
<td>1800x1800</td>
<td>3.43</td>
<td>5.34</td>
<td>7.64</td>
</tr>
<tr>
<td>1900x1900</td>
<td>3.41</td>
<td>5.31</td>
<td>7.60</td>
</tr>
<tr>
<td>2000x2000</td>
<td>3.40</td>
<td>5.28</td>
<td>7.56</td>
</tr>
</tbody>
</table>

Table 2: Practical speed up ratio for images with different sizes using MATLAB.

<table>
<thead>
<tr>
<th>Image size</th>
<th>Speed up ratio (n=20)</th>
<th>Speed up ratio (n=25)</th>
<th>Speed up ratio (n=30)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100x100</td>
<td>7.88</td>
<td>10.75</td>
<td>14.69</td>
</tr>
<tr>
<td>200x200</td>
<td>6.21</td>
<td>9.19</td>
<td>13.17</td>
</tr>
<tr>
<td>300x300</td>
<td>5.54</td>
<td>8.43</td>
<td>12.21</td>
</tr>
<tr>
<td>400x400</td>
<td>4.78</td>
<td>7.45</td>
<td>11.41</td>
</tr>
<tr>
<td>500x500</td>
<td>4.68</td>
<td>7.13</td>
<td>10.79</td>
</tr>
<tr>
<td>600x600</td>
<td>4.46</td>
<td>6.97</td>
<td>10.28</td>
</tr>
<tr>
<td>700x700</td>
<td>4.34</td>
<td>6.83</td>
<td>9.81</td>
</tr>
<tr>
<td>800x800</td>
<td>4.27</td>
<td>6.68</td>
<td>9.60</td>
</tr>
<tr>
<td>900x900</td>
<td>4.31</td>
<td>6.79</td>
<td>9.72</td>
</tr>
<tr>
<td>1000x1000</td>
<td>4.19</td>
<td>6.59</td>
<td>9.46</td>
</tr>
<tr>
<td>1100x1100</td>
<td>4.24</td>
<td>6.66</td>
<td>9.62</td>
</tr>
<tr>
<td>1200x1200</td>
<td>4.20</td>
<td>6.62</td>
<td>9.57</td>
</tr>
<tr>
<td>1300x1300</td>
<td>4.17</td>
<td>6.57</td>
<td>9.53</td>
</tr>
<tr>
<td>1400x1400</td>
<td>4.13</td>
<td>6.53</td>
<td>9.49</td>
</tr>
<tr>
<td>1500x1500</td>
<td>4.10</td>
<td>6.49</td>
<td>9.45</td>
</tr>
<tr>
<td>1600x1600</td>
<td>4.07</td>
<td>6.45</td>
<td>9.41</td>
</tr>
<tr>
<td>1700x1700</td>
<td>4.03</td>
<td>6.41</td>
<td>9.37</td>
</tr>
<tr>
<td>1800x1800</td>
<td>4.00</td>
<td>6.38</td>
<td>9.32</td>
</tr>
<tr>
<td>1900x1900</td>
<td>3.97</td>
<td>6.35</td>
<td>9.28</td>
</tr>
<tr>
<td>2000x2000</td>
<td>3.94</td>
<td>6.31</td>
<td>9.25</td>
</tr>
</tbody>
</table>