# Parallel Algorithm Of Mathematical Model For Fuzzy Neural Network Used In Image Deblurring

**Associate Pro. Dr. Sadiq A. Mehdi**
**Al-Mustansiriya University**

**Maytham A. Ali**
**Baghdad College of Economic Sciences University**

**Mohammed A. Subhi**
**Baghdad College of Economic Sciences University**

## Abstract:

A mathematical model for parallel algorithm was proposed for the image deblurring using Fuzzy neural network recognition system. The proposed mathematical model was used to suggest a new parallel adaptive neural learning method used to increase the recognition accuracy and decrease time.
To solve image blurring problem, a new de-blurring strategy was proposed in order to reduce or to remove the blur in the images. The proposed strategy was suggested to use the Least Square interpolation controlled by the proposed parallel learning fuzzy-neural network to select the blurred pixels. The proposed strategy gives good results for fully and partially blurred region in images.

**Keyword:** Mathematical Models Application, Parallel Algorithm, Fuzzy neural network, Image deblurring.

## المستخلص:

النموذج الرياضي المقترح للخوارزمية المتوازية قد صمم لرفع الضبابية ( التغويش) باستخدام نظام الشبكات العصبونية المضببة. النموذج الرياضي استخدم لأقتراح طريقة جديدة متكيفة متوازية لتعليم الشبكات العصبية وبهذا تستخدم لزيادة جودة الاسترجاع وخفض وقته. لحل مشكلة ازالة الغواش يقترح البحث استراتيجية جديدة من خلال طريقة استيفاء المربع الاصغر المستخدم في تعلم الشبكات العصبونية لاختيار النقاط المغوشة في الصورة. الاستراتيجية المقترحة تظهر نتائج جيدة من خلال تطبيقها على الصور المغوشة جزئيا او كليا.

## 1. Introduction

The finite element method has proven to be an effective tool in the numerical solution of many scientific and engineering applications. An implementation of the method depends on a spatial decomposition of the computational domain into a union of simple geometric elements and a corresponding computational mesh. The task of generating the mesh can be one of the most time-consuming aspects of the entire solution process. If the geometry is complex, automatic mesh generation tools can be used to facilitate the decomposition. Unfortunately, meshes generated in this way can contain poorly shaped or distorted elements which result in numerical difficulties during the solution process. For example, it has been shown that as element angles become too large the discretization error in the finite element solution is increased and as angles become too small the condition number of the element matrix is increased. Thus, for meshes containing distorted elements, the numerical solution is more difficult to compute and the numerical approximation is less accurate (Freitag, Jones, & Plassmann, 1995).

One approach used to correct these problems is to adjust the mesh point locations in such a way that element distortion is reduced and the overall quality of the mesh is improved. Several mesh smoothing methods and algorithms have been proposed to perform this adjustment on sequential computers. However, there are many large-scale applications, for example the “grand challenge” problems, that require additional memory capacity and computational power of a massively parallel machine for their solution. The design philosophy of the parallel algorithm follows three basic tenets: (1) the algorithm must be as effective as the best sequential algorithm; (2) the algorithm must have a provably fast parallel running time bound; and (3) the software implementation of the algorithm must be efficient and portable (Freitag, Jones, & Plassmann, 1995).
A mathematical model for parallel algorithm was proposed and applied to the speech recognition operation.

## 2. Models of Parallel Computation

One very basic fact that applies to parallel computation, regardless of how it is implemented, as the following:
CLAIM 1.1: Suppose the fastest sequential algorithm for doing a computation with parameter *n* has execution time of *T(n)*. Then the fastest parallel algorithm with *m* processors (each comparable to that of the sequential computer) has execution time¸ T(n)/m (Smith, 1993).

The idea here is: If you could find a faster parallel algorithm, you could execute it sequentially by having a sequential computer simulate parallelism and get a faster sequential algorithm. This would contradict the fact that the given sequential algorithm is the fastest possible. We are making the assumption that the cost of simulating parallel algorithms by sequential ones is negligible. This claim is called the “Principle of Unitary Speedup”.
As usual, the parameter n represents the relative size of the instance of the problem being considered. For instance, if the problem was that of sorting, n might be the

number of items to be sorted and T(n) would be O(n lg n) for a sorting algorithm based upon comparisons.

As simple as this claim is, it is a bit controversial. It makes the tacit assumption that the algorithm in question is deterministic. In other words, the algorithm is like the usual idea of a computer program, it performs calculations and makes decisions based on the results of these calculations.

There is an interesting area of the theory of algorithms in which statement 1.1 is not necessarily true, this is the theory of randomized algorithms. Here, a solution to the problem may involve making random "guesses" at some stage of the calculation. In this case, the parallel algorithm using m processors can run faster than m× the speed of the sequential algorithm ("Super-unitary speedup"). This phenomenon occurs in certain problems in which random search is used, and most guesses at a solution quickly lead to a valid solution, but there are a few guesses that execute for a long time without producing any concrete results (Smith, 1993).

Modeling parallel computations is more complicated than modeling sequential computations because in practice parallel computers tend to vary more in organization than do sequential computers. As a consequence, a large portion of the research on parallel algorithms has gone into the question of modeling, and many debates have raged over what the "right" model is, or about how practical various models are (Blelloch & Maggs, 2010).

With the development of commercially-available parallel processing, some new terms have come into common use:

**Procedure-level parallelism**: This represents parallel programming for a computer that has a relatively small number of processors. Since the number of processors is small, each processor usually does a large chunk of the computation.

The different processors wind up executing procedures in parallel, so this style of programming is called procedure-level parallelism. It has the flavor of concurrent programming (where there is real concurrency) and many standard concurrent programming constructs are used, like semaphores, monitors, and message passing.

**Data-level parallelism**: This represents the style of parallel programming that is emphasized. It is used when there is a large number of a processor on a machine that may be SIMD or MIMD. The name 'data-level parallelism' is derived from the idea that the number of processors is so great that the data can be broken up and sent to different processors for computations. With a large number of processors you could send each iteration of the loop to a separate processor. In contrast to procedure-level parallelism, where you broke the code up into procedures to be fed to different processors, here you break up the data and give it to different processors.

These terms are not particularly meaningful — they are only valid if one does parallel programming in a certain way that is closely related to ordinary sequential programming (i.e. the terms arose when people tried to parallelize sequential programs in a fairly straightforward way). The terms are widely used, however (Smith, 1993).

## 3. Neural Network

Many NN models and learning algorithms have been proposed. Typical network structures include feedback and feed-forward NNs. Learning algorithms are categorized into supervised learning and unsupervised learning. This section provides an overview of these models and algorithms (Takagi, 1997).

The feed-back networks are NNs that have connections between network outputs and some or all other neuron units. Certain unit outputs in the figure are used as activated inputs to the network, and other unit outputs are used as network outputs.

Due to the feed-back, there is no guarantee that the networks become stable. Some networks converge to one stable point, other networks become limit-cycle, and others become chaotic or divergent. These characteristics are common to all non-linear systems which have feed-back.

To guarantee stability, constraints on synaptic weights are introduced so that the dynamics of the feed-back NN is expressed by the Lyapunov function. Concretely, a constraint of equivalent mutual connection weights of two units is implemented. The Hopfield network is one such NNs. It is important to understand two aspects of the Hopfield network: (1) Synaptic weights are determined by analytically solving constraints not by per-forming an iterative learning process. The weights are fixed during the Hopfield network runs. (2) Final network outputs are obtained by running feed-back networks for the solutions of an application task. Another type of NN which is compared with the feed-back type is a feed-forward type. The feed-forward network is a filter which outputs the processed input signal. Several algorithms determine synaptic weights to make the outputs match the desired result.

Supervised learning algorithms adjust synaptic weights using input–output data to match the input–output characteristics of a network to desired characteristics.

The most frequently used algorithm, the backpropagation algorithm, is explained in detail in the next section. Unsupervised learning algorithms use the mechanism that changes synaptic weight values according to the input values to the network, unlike supervised learning which changes the weights according to supervised data for the output of the network. Since the output characteristics are determined by the NN itself, this mechanism is called *self-organization*.

Hebbian learning and competitive learning are representative of unsupervised learning algorithms (as shown in Fig.1). A Hebbian learning algorithm increases a weight, *wi*, between a neuron and an input,*xi*, if the neuron, *y*, fires.

$$\Delta wi = ayxi \qquad \ldots(1)$$

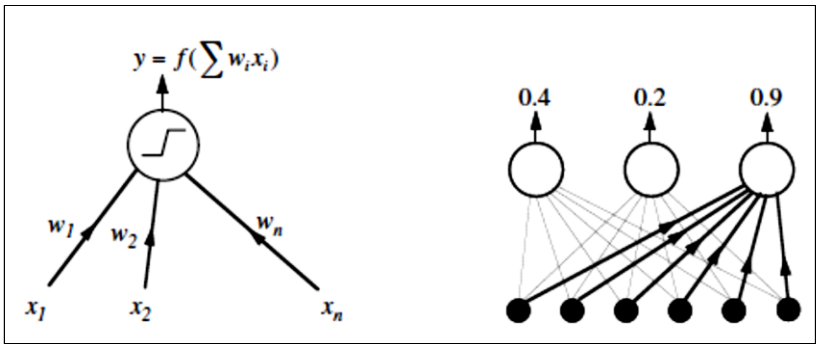
where *a* is a learning rate. Any weights are strengthened if units connected with the weights are activated.

Weights are normalized to prevent an infinite increase in weights. Competitive learning algorithms modify weights to generate one unit with the greatest output. Some variations of the algorithm also modify other weights by lateral inhibition to suppress the outputs of other units whose outputs are not the greatest. Since only one unit becomes active as the winner of the competition, the unit or the network is called a *winner-take-all* unit or network. Kohonen’s self-organization feature map, one of the most well-known competitive NNs, modifies the weights connected to the winner-take-all unit as:

$$\Delta wi = a(xi - wi) \qquad \textbf{... (2)}$$

Where the sum of input vectors is supposed to be normalized as 1,

**Learning algorithm:** Signal flow of a feed-forward NN is unidirectional from input to output units. Figure 1, 2 shows a numerical example of the data flow of a feed-forward NN.



(a) (b)

Figure 1: (a) Hebbian learning algorithms strength weight, *wi* when input, *xi* activates a neuron, *y*. (b) Competitive learning algorithms strength only weights connected to the unit whose output is the biggest

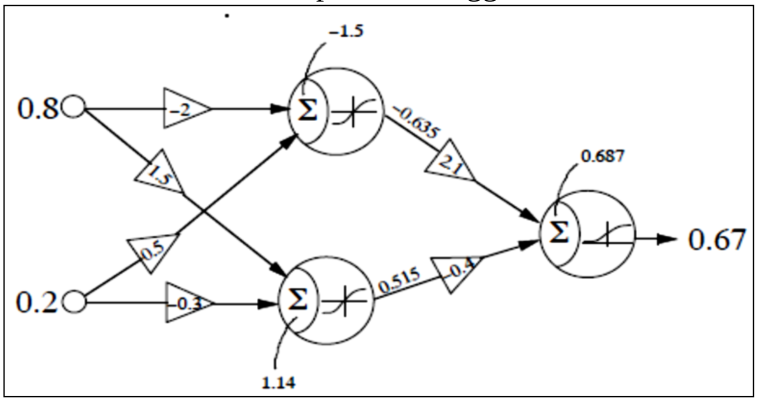


Figure 2: Example data flow in a simple feed-forward neural network

One of the most popular learning algorithms which iteratively determine the weights of the feed-forward NNs is the backpropagation algorithm. A simple learning algorithm that modifies the weights between output and hidden layers is called a delta rule. The backpropagation algorithm is an extension of the delta rule that can train the weights, not only between output and hidden layers but also hidden and input layers.

Let E be an error between the NN outputs, $v^3$, and supervised data, y. The number at superposition means the layer number. Since NN outputs are changed when synaptic weights are modified, the E must be a function of the synaptic weights w:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{N_k}(v_j^3 - y_j)^2. \qquad \textbf{... (3)}$$

Supposed that, in Fig.3, the vertical axis is *E* and the *x*1 ... *xn* axes are the weights, *w*1 ... *wn*. Then, NN learning is to find the global minimum coordinate in the surface of the figure. Since *E* is a function of **w**, the searching direction of the smaller error point is obtained by calculating a partial differential. This technique is called the gradient method, and the steepest decent method is the base of the backpropagation algorithm. The searching direction, $\mathbf{g} = -\partial E(\mathbf{w})/\partial \mathbf{w}$, and modification of weights is given as $\Delta\mathbf{w} = \varepsilon\mathbf{g}$. From this equation, we finally obtain the following back-propagation algorithm.

$$\Delta w_{i,j}^{k-1,k} = -\epsilon d_j^k v_i^{k-1}$$
$$d_j^k = \begin{cases} (v_j^3 - y_j)\frac{\partial f(U_j^k)}{\partial U_j^k} & \text{for output layer} \\ \sum_{h=1}^{N_{k+1}} d_j^{k+1} w_{i,h}^{k,k+1} \frac{\partial f(U_j^k)}{\partial U_j^k} & \text{for hidden layer} \end{cases} \quad \ldots (4)$$

Where $w^{k-1,k}{}_{i,j}$ is the connection weight between the *i*-th unit in the $(k-1)$-th layer and the *j*-th unit in the *k*-th layer, and $U^k{}_j$ is the total amount of input to the *j*-th unit at *k*-th layer. To calculate $d^k{}_j$, $d^{k+1}{}_j$ must be previously calculated.
Since the calculation must be conducted in the order of the direction from the output layer to input layer, this algorithm is named the *backpropagation* algorithm.
When a sigmoidal function is used for the characteristic function, *f*(x), of neuron units, the calculation of the algorithm becomes simple

$$f(x) = \frac{1}{1+\exp^{-x+T}}$$
$$\frac{\partial f(x)}{\partial x} = (1 - f(x))f(x) \quad \ldots (5)$$

**4. Neural Networks** (Othman, & Riadh, 2008)

Neural networks model some aspects of the human brains, where thinking process is achieved in synaptic connections between neurons. The structure of the network is layered and capable of high parallelism. Neural networks are useful in classification, function approximation and generally in complex problems, which do not require accurate solution.
Neural networks must be taught before they can be used, which corresponds to how humans learn. A neural network consists of units (processors, nodes) that are interconnected with several other such units; they function independently on the input they are given and their local data. Usually all of the units in a network are homogenous, but also heterogeneous networks exist.

**5. The Scaly Neural Network Architecture**

A problem with fully connected neural networks is their size. When it comes to the practical implementation of neural networks size becomes an important factor. In the case of computer simulation the problem comes with the large computational cost. The larger and more complex the network the longer it takes to train and once trained it takes longer for the network to perform its recognition task.
Fig.9: shows an example of a neural network where scaly architecture has been applied between the input layer and the hidden layer. This is the approach adopted for the work

here since the localized structure of the input zones is somewhat analogous to the Cochlear processing, which occurs, in the human ear. A preliminary investigation into the ability of a scaly architecture neural network was carried out by A.D. Smith at the University of Newcastle upon Tyne. Smith' work suggested that further investigation of this network was required to better determine the effect of changing the parameters of the network.

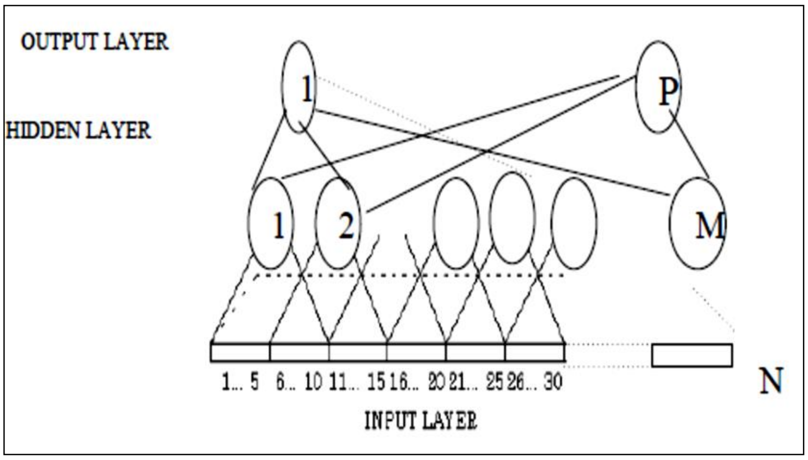


Fig. 9: Scaly Neural Networks

In (Othman, & Riadh, 2008) the scaly neural network architecture that is a scaly connected network requires an input layer of 780 input neurons to accommodate the 65 input feature vectors, each contains 12 coefficients. For the scaly architecture, the number of frames in a zone is taken as 10 frames with an overlap of 5 frames so that the number of nodes required in hidden layer is 132 nodes; the number of output nodes is 11 nodes.

## 5.1 Neural Networks for Training and Testing

The input data is processed and presented to the network so that successive feature vectors or frames are presented to the network inputs, each coefficient of the feature vectors being presented to one of the input nodes.

The data is used in the speaker dependent training and testing involves using four utterances spoken by each speaker for training and the other four utterances for testing. The mean length of an utterance over the entire data set is calculated and found to be 65 frames.

## 6. The Proposed Mathematical Model

The following description of the proposed development for the speech recognition in (Othman, & Riadh, 2008). The developing will be described as a mathematical model for parallel algorithm changing the neural network learning ways controlled by Fuzzy rules.

Speech -independent voice recognition is a large and difficult classification problem.

**A neuro-fuzzy classifier ( NFC )**

The mathematical model of the NFC is based on a multi-layer, with alterations in operation to accommodate fuzzy data (Tebelskis, 1995). The NFC concept can be summarized as a three-step procedure. First, the data is preprocessed to assign appropriate membership function values. Second, the data is processed via $\alpha$-cuts. Last, the fuzzy output is defuzzified. Further details are divided between formal description and procedural techniques, as shown below.

**Formal Description**

The NFC is based on the multilayer and can be formally defined by seven-parameters,

$$NFC = (\mathbf{F}_n, \mathbf{F}_m, \tilde{\mathbf{I}}, \tilde{\mathbf{O}}, \mathbf{A}, \mathbf{L}, \mathbf{f}) \qquad \textbf{... (6)}$$

**Where**:

$\mathbf{F}_n$ is the input field, an ordered array of neurons of size n × 1,

$\mathbf{F}_m$ is the output field, an ordered array of neurons of size m × 1,

$\tilde{\mathbf{I}}$ is the fuzzy input, arranged as an interval vector, $(X_1, X_2, ..., X_n) \in \Re^{2n}$, where $X_i$ corresponds to the fuzzy input (represented as an interval) to neuron i in the input field, $F_n$,

$\tilde{\mathbf{O}}$ is the fuzzy output, arranged as an interval vector, $(Y_1, Y_2, ..., Y_m) \in \Re^{2m}$, where $Y_i$ corresponds to the output (represented as an interval) from neuron i in the output field, $F_m$,

$\mathbf{A}: \Re^{2n} \rightarrow \Re^{2m}$ is the association function,

$\mathbf{L} \subseteq \mathbf{A}$ is the set of learned associations, and

$f: \Re^{2k} \rightarrow \Re^{2}$ is the neurons' activation function.

The basic structure of the NFC relies on neurons that have weights and an activation function that are found. The inputs, however, are fuzzy and represented by intervals based on a standard level set. That is, each $\alpha$-cut for pattern p is represented by the interval vector $\mathbf{X}_p = (X_{p1}, X_{p2}, \ldots\ldots, X_{pn})$

**Where:**

$$\mathbf{X}_{\mathbf{pi}} = [\mathbf{x}^{\mathbf{L}}_{\mathbf{pi}}, \mathbf{x}^{\mathbf{U}}_{\mathbf{pi}}] \qquad \textbf{... (7)}$$

Indicate the lower and upper limits of the interval. The neuronal function, being crisp, then processes the interval vector via interval mathematics and produces an interval vector as output. Specifically, the summation of weighted inputs for neuron j is carried out as:

$$Net^{L}_{pj} = \sum_{\substack{i \\ w_{ji} \geq 0}} w_{ji} Y^{L}_{pi} + \sum_{\substack{i \\ w_{ji} < 0}} w_{ji} Y^{U}_{pi} + \theta_j \qquad \textbf{... (8)}$$

And:

$$Net^{U}_{pj} = \sum_{\substack{i \\ w_{ji} \geq 0}} w_{ji} Y^{U}_{pi} + \sum_{\substack{i \\ w_{ji} < 0}} w_{ji} Y^{L}_{pi} + \theta_j \qquad \textbf{... (9)}$$

where $\theta_j$ is the bias. The output, then, is calculated as:

$$\begin{aligned} Y_{pj} &= [Y_{pj}^{L}, Y_{pj}^{U}] \\ &= [f(\mathrm{Net}_{pj}^{L}), f(\mathrm{Net}_{pj}^{U})] \end{aligned} \qquad \dots (10)$$

The learning algorithm is based on back-propagation with momentum. There is an error calculated and back-propagated in order to modify the weights. Specifically, the error is computed as the difference between the target output, $\mathbf{t}_p$ (for pattern p), and the actual output, $\mathbf{Y}_p$:

$$E_p = \max\{\frac{1}{2}(t_{pj} - Y_{pj})^2 , Y_{pj} \in Y_p\} \qquad \dots (11)$$

where

$$(\,tpj - Ypj\,) = \begin{cases} (t_{pj} - Y_{pj}^{L}), & \text{if } t_p = 1 \\ \\ (t_{pj} - Y_{pj}^{U}), & \text{if } t_p = 0. \end{cases} \qquad \dots (12)$$

The learning rule is based with modifications for interval mathematics. The change in any weight (in any layer) is:

$$\Delta w_{ji}(t+1) = \eta(-\frac{\partial E_p}{\partial w_{ji}}) + \alpha \Delta w_{ji}(t) \qquad \dots (13)$$

For units in the output layer, the calculation of $\partial E_p / \partial w_{ji}$ is straightforward, and can be thought of as four cases based on the value of target output and weight. Note that in the four equations the value of j in the subscript is fixed (to the output neuron that had the maximum error). In the first case, the $\mathbf{t_p = 1}$**, and** $\mathbf{w_{ji} \geq 0}$.
The crisp weights are obtained via a training phase based on back-propagation with momentum.

## 7. The Least Squares Estimation Method

The method of least squares is a standard approach to the approximate solution of over determined systems, i.e. sets of equations in which there are more equations than unknowns. "Least squares" means that the overall solution minimizes the sum of the squares of the errors made in solving every single equation (Hussien & Naif, 2010).

The most important application is in data fitting. The best fit in the least-squares sense minimizes the sum of squared residuals, a residual being the difference between an observed value and the fitted value provided by a model.

Least squares problems fall into two categories: linear least squares and nonlinear least squares, depending on whether or not the residuals are linear in all unknowns. The linear least-squares problem occurs in statistical regression analysis; it has a closed-form solution. The non-linear problem has no closed solution and is usually solved by iterative refinement; at each iteration the system is approximated by a linear one, thus the core calculation is similar in both cases.

The objective consists of adjusting the parameters of a model function to best fit a data set. A simple data set consists of n points (data pairs) $(x_i, y_i)$, i = 1, ..., n, where $x_i$ is an independent variable and $y_i$ is a dependent variable whose value is found by observation. The model function has the form f(x, β), where the m adjustable parameters are held in the vector $\beta$. The goal is to find the parameter values for the model which "best" fits the data. The least squares method finds its optimum when the sum, S, of squared residuals is minimum.

$$S = \sum_{i=1}^{n} r_i^2 \quad \text{…. (14)}$$

A residual is defined as the difference between the value predicted by the model and the actual value of the dependent variable

$$ri = f(xi,\beta) - yi. \quad \text{…. (15)}$$

**8. The Proposed Least Squares Interpolation Filter**

In this section, the Least Square filter design will explain. This filter was proposed to repaint image depend on the least square calculation method. The result from this filter will used to remove or reduce the blurring in the blurred images. The main equation of this proposed filter is develop to the eq. (19) and (20) applied to the suggested mask. The Least square developed equation is:

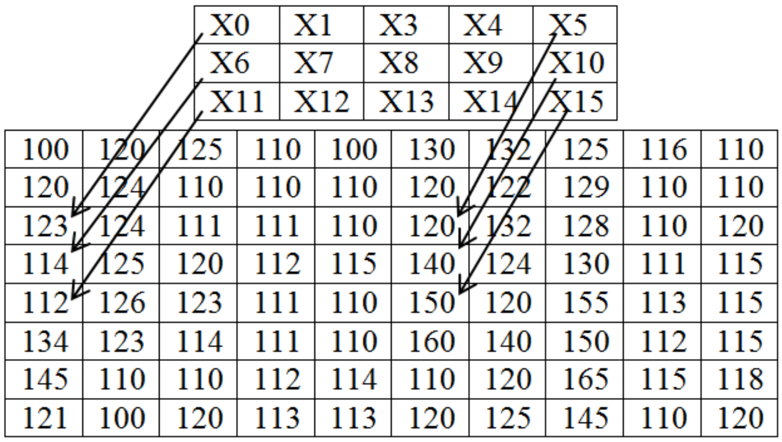
$$S = \sum_{i=1}^{n} r_i^2 \quad \text{…. (16)}$$

$$r_i = ((f(x_i) - y_i) + (M-y_i))/2 \quad \text{…(17)}$$

where M is the average pixels for the suggested mask.

The Least Squares interpolation resulted values that used to replace the selected pixels in the original image depend on the net of neighbors pixels effect. The replacing operation will covered all pixels on the suggested mask.

The suggested mask design to gives the best pixels effect calculation. The suggested mask contains three rows 5 image pixels (3x5). Figure (2) shows an example of the design mask. The mask will apply to whole image until deblurring operation complete.



| | | | | |
|---|---|---|---|---|
| X0 | X1 | X3 | X4 | X5 |
| X6 | X7 | X8 | X9 | X10 |
| X11 | X12 | X13 | X14 | X15 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 120 | 125 | 110 | 100 | 130 | 132 | 125 | 116 | 110 |
| 120 | 124 | 110 | 110 | 110 | 120 | 122 | 129 | 110 | 110 |
| 123 | 124 | 111 | 111 | 110 | 120 | 132 | 128 | 110 | 120 |
| 114 | 125 | 120 | 112 | 115 | 140 | 124 | 130 | 111 | 115 |
| 112 | 126 | 123 | 111 | 110 | 150 | 120 | 155 | 113 | 115 |
| 134 | 123 | 114 | 111 | 110 | 160 | 140 | 150 | 112 | 115 |
| 145 | 110 | 110 | 112 | 114 | 110 | 120 | 165 | 115 | 118 |
| 121 | 100 | 120 | 113 | 113 | 120 | 125 | 145 | 110 | 120 |

Fig.(2) Example of The Least Squares Interpolation Mask

As shown in Fig.(2), the mask points to selected named by x0, x1, x2, x3, …x15. These selected pixels will replaced by the resulted values from the Least Square calculation. (Note: the replacing operation will control by the neural network).

## 9. The Image Deblurring System

Restoration or deblurring average blur from images is a very difficult problem to resolve. In this research we describe how to use the proposed parallel learning fuzzy-neural network algorithm in the application like image deblurring system. Many strategies can be used for solving such image blurring problems. This proposed strategy is used to evaluate the proposed filter to de-blur the image (explained in section 7) which is controlled by fuzzy neural network as explained in section 6 the aforementioned filter uses the proposed mathematical parallel leaning model. The structure of the neural network used in this proposed system is explained in the following section.

As explained earlier in section 7, the proposed filter was designed using the principles and theory of the least polynomial interpolation to estimate the pixels values to replace in the mask window of the filter.

Image will be divided into blocks. For each block mask window, the proposed system will use the proposed parallel learning fuzzy-neural network to test each pixel in the block, if the pixel is blurred or not. The proposed parallel learning fuzzy-neural network was learned to cover three types of the blurring (Gaussian, rectangular, and the motion linear uniform horizontal).

In the same time, the proposed system applied the proposed Least Square filter to calculate the new replacement pixels values. If the selected pixel is blurred it assigned with the blur type, the proposed system will replace the pixel by the proposed filter result. Else, the pixel will be left without replacement. The proposed filter and neural network will apply to whole image. Finally, the proposed system recomposes the resulted image and calculates the RMSE measure to check the variation between the original image and resulted image.

The main proposed system steps are as follows:

- Partitioning the image into a window of 3x5 pixels as blocks and calculate the average of each block.
- Apply neural network on the image blocks.
- If neural decision is blurred block then:
    - Apply Least Square filter on the image mask.
    - Replace with result of the proposed Least Square filter.
    - Calculate RMSE
    - Save the de-blurred image.

Figure 3 shows the block diagram of the proposed system block:

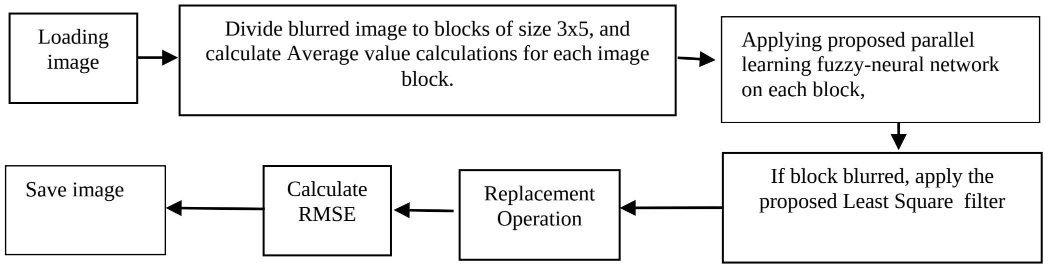

Loading image
Divide blurred image to blocks of size 3x5, and calculate Average value calculations for each image block.
Applying proposed parallel learning fuzzy-neural network on each block,
If block blurred, apply the proposed Least Square filter
Replacement Operation
Calculate RMSE
Save image

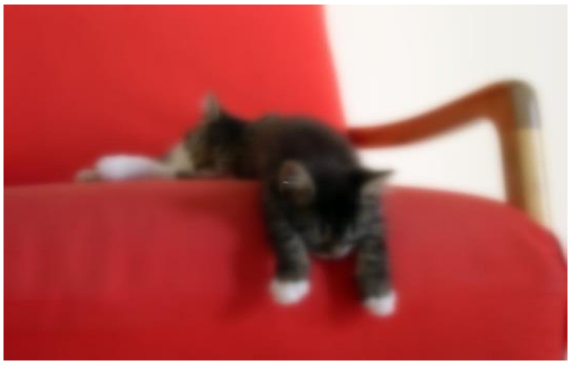
Fig.(3) the block diagram of the proposed system

## 10. Experimental Results & Discussions

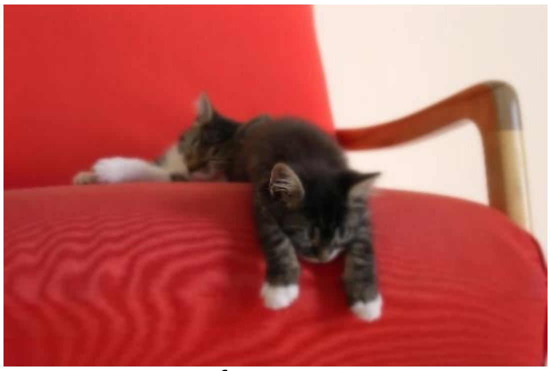
In this research a Least Square interpolation calculation was used with proposed parallel learning fuzzy-neural network based on parallel-valued neurons learning to deblur images with a full blurring or partial blurring. The proposed filter and neural network give very good results in removing the fully/partially blurring from images. Also, a good RMSE values are resulted. Fig(5) shows some proposed system results. The RMSE of the different samples deblurring by proposed Least Square filter is compared in Table 1.

Table 1 RMSE Results from applies the proposed deblurring system

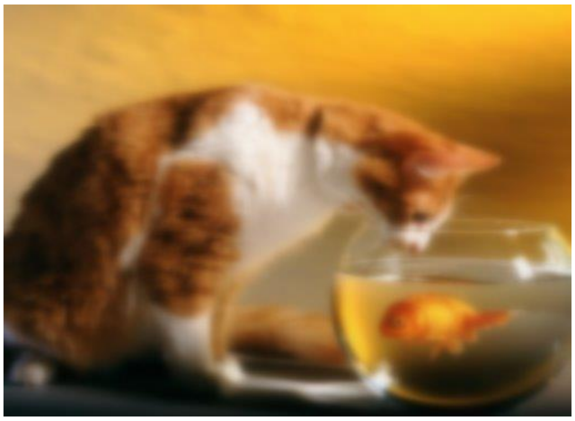
| Sample name | RMSE | Size |
|---|---|---|
| test1 | 39.89 | 640x480 |
| test2 | 43.90 | 600x800 |
| test3 | 44.50 | 600x800 |
| test4 | 46.85 | 512x 512 |
| test5 | 47.37 | 512x512 |
| test6 | 47.23 | 600x800 |



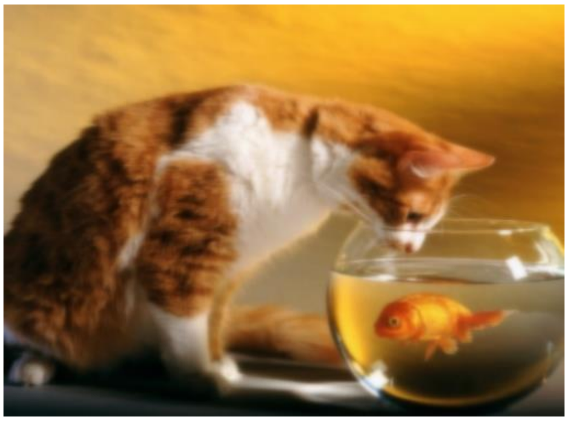
Before



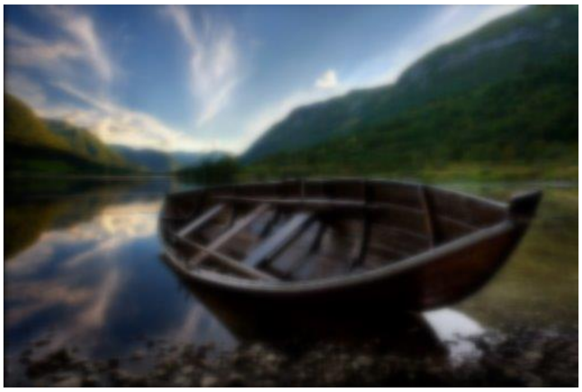
After



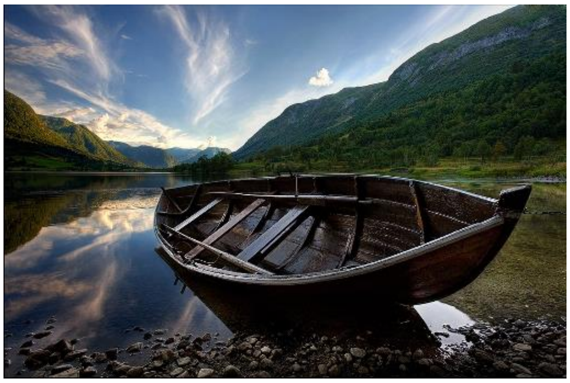
Before



After



Before



After

Fig(5) Some results of the proposed Least Square filter.

## References

Blelloch, G., & Maggs, B. (2010). *Algorithms and theory of computation handbook.* Chapman & Hall/CRC.

Freitag, L., Jones, M., & Plassmann, P. (1995). An Efficient Parallel Algorithm for Mesh Smoothing. *4th International Meshing Roundtable* (pp. 47-58). Sandia National Laboratories.

Hussien, K., & Naif, J. (2010). *The Least Squares Interpolation Polynomial for Images.*

Othman, , A., & Riadh, M. (2008). *Speech Recognition Using Scaly Neural Networks.* World Academy of Science, Engineering and Technology.

Smith, J. (1993). *The Design and Analysis of Parallel Algorithms.* Oxford University Press.

Takagi, H. (1997). *Introduction to Fuzzy Systems, Neural Networks, and Genetic Algorithms.* Kyushu Institute of Design.

Tebelskis, J. (1995). *Speech Recognition using Neural Networks, Ph.D thesis.* Pittsburgh, Pennsylvania: School of Computer Science, Carnegie Mellon University.