



Jaypee University of Information Technology
Solan (H.P.)

LEARNING RESOURCE CENTER

Acc. Num. SP06086 Call Num:

General Guidelines:

- ◆ Library books should be used with great care.
- ◆ Tearing, folding, cutting of library books or making any marks on them is not permitted and shall lead to disciplinary action.
- ◆ Any defect noticed at the time of borrowing books must be brought to the library staff immediately. Otherwise the borrower may be required to replace the book by a new copy.
- ◆ The loss of LRC book(s) must be immediately brought to the notice of the Librarian in writing.

Learning Resource Centre-JUIT

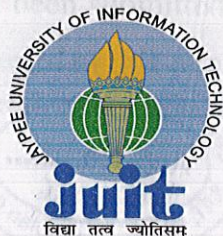


SP06086

**A DETAILED STUDY AND IMPLEMENTATION
OF IMAGE PROCESSING TECHNIQUES USING
PARTICLE SWARM OPTIMIZATION (PSO)
AND CELLULAR NEURAL NETWORKS (CNN)**

By

TUSHAR JAISWAL - 061304



MAY-2010

**Submitted in partial fulfillment of the Degree of Bachelor of
Technology**

**DEPARTMENT OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY**

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-
WAKNAGHAT**

CERTIFICATE

This is to certify that the work entitled, "**A Detailed Study and Implementation of Image Processing Techniques Using Particle Swarm Optimization (PSO) and Cellular Neural Networks (CNN)**" submitted by **Tushar Jaiswal – 061304** in partial fulfillment for the award of degree of Bachelor of Technology in **Computer Science Engineering** of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.



Dr. Rajesh Siddavatam, PhD

Assistant Professor

Department of Computer Science and Information Technology


Jaypee University of Information Technology

Waknaghat

ACKNOWLEDGMENTS

I would like to acknowledge **Dr. Rajesh Siddavatam** for his gracious support and guidance in this project. I appreciate his immense help and his feedbacks which helped me in developing the project so successfully.

I would also like to thank **Mr. Praveen Kumar Tripathi** for providing me with valuable help in learning Particle Swarm Optimization (PSO). I am grateful to him for helping me in understanding PSO. It is because of his immense help that I was able to use PSO in the project.


19-05-2010

Tushar Jaiswal

061304

PROBLEM STATEMENT

The project necessitated a thorough study and implementation of Image Processing Techniques. As such the main focus of the project has been to deal with the problem of Image Noise Cancellation.

Unwanted noise is introduced in images during the process of Image Acquisition and Image Transmission. This can be because of faulty sensors or errors in the communication channel.

Before we subject the acquired image to further image processing like Image Segmentation, Edge Detection, Object Recognition, Pattern Recognition etc. it is quintessential that the image be freed from noise so that further processing of the image does not give erroneous results.

Various filters have been designed and are being designed for noise removal from images. Median based filters and its variations have been used extensively for noise cancellation.

Through this project I seek to develop new and better filters and algorithms for Image Noise Cancellation. I also seek to study and improve upon some previous methods for Image Noise Cancellation.

OBJECTIVE AND SCOPE OF THE PROJECT

The main objective of the project is to devise new and better methods for Image Processing tasks such as Image Noise Cancellation.

The project also aims at studying the existing algorithms in detail, implementing them and to develop ways to improve the methods by removing their shortcomings.

This will be done with the help of two technologies Particle Swarm Optimization (PSO) and Cellular Neural Networks (CNN).

PSO is an optimization technique which aims at improving the results of some process. Through the use of PSO the project aims to improve the Noise Cancellation process.

CNN is a technique in which the system learns and evolves through the crossover and mutation of data. It thus aims to get the best results by learning from templates designed for the specific application.

The project will thus eventually develop new and better algorithms for the Image Processing tasks such as Image Noise Cancellation which can be put to effective use.

The algorithms can be used by researchers and people from the Image Processing Industry alike.

The Graphical User Interface (GUI) developed can be easily used to implement the algorithms and to learn the methods for Image Processing.

METHODOLOGY

The project will be done through a three tiered approach:

- Study of existing algorithms and their implementation
- Developing new and better algorithm and comparing them to other algorithms by implementing them
- Developing Graphical User Interface (GUI) for the developed algorithms and the algorithms studied

The study of existing algorithms will be done to get a strong background of the problems faced in image processing and to learn the approaches to solving them. In this way I can broaden my knowledge base of Image Processing. By implementing the algorithms I will hammer home the concepts that would have been learned.

After developing a sound base I'll devise new methods for Image Processing tasks. It will be seen that the developed algorithms are in fact better than at least many current good methods by implementing them and comparing them with the other existing methods.

Finally all the algorithms developed will be compiled and GUI will be developed for using them. This will enable the end user to easily run and learn the developed algorithms. Also GUI will be developed for some algorithms which have been studied so that their effectiveness can be seen. It will also be then easier to understand those algorithms and work further after learning.

RESOURCES AND LIMITATIONS

The main resources used in the project are:

- Research Material from IEEE Xplore, ACM, Springerlink, Science Direct etc.
- MATLAB
- MATLAB Documentation
- Digital Image Processing by Rafael C. Gonzalez
- Digital Image Processing Using Matlab by Rafael C. Gonzalez
- Wikipedia
- Tutorials and other study material from the internet

The limitation of the project is:

- Nil

TABLE OF CONTENTS

| | |
|--|-----------|
| 1. Acknowledgments..... | i |
| 2. Problem Statement..... | ii |
| 3. Objective and scope of the project..... | iii |
| 4. Methodology..... | iv |
| 5. Resources and limitations..... | v |
| 6. Table of Contents..... | vi-vii |
| 7. List of Figures..... | viii-ix |
| 8. List of Tables..... | x |
| 9. List of Abbreviations..... | xi |
| 10. Abstract..... | xii |
| 11. Chapter 1 – Noise In Image Processing..... | 1-5 |
| 1.1 Noise..... | 2 |
| 1.2 Salt and Pepper Noise..... | 3 |
| 1.3 Evaluation Measures..... | 3 |
| 12. Chapter 2 – Lifting Scheme..... | 6-25 |
| 2.1 Second Generation Wavelets..... | 6 |
| 2.2 Lifting Scheme..... | 7 |
| 2.3 Applications..... | 9 |
| 13. Chapter 3 – Diminshing Population Particle Swarm Optimization (DP-PSO) | 26-44 |
| 3.1 Particle Swarm Optimization..... | 26 |
| 3.2 Diminshing Population Particle Swarm Optimization..... | 32 |
| 3.3 Application..... | 33 |
| 14. Chapter 4 – Cellular Neural Netwroks (CNN) | 45-49 |
| 4.1 Introduction..... | 45 |
| 4.2 A Simple Example..... | 48 |

| | |
|---|--------------|
| 15. Chapter 5 – Project Simulator..... | 50-55 |
| 5.1 Building Graphical User Interfaces using GUIDE in Matlab..... | 50 |
| 5.2 The Simulator..... | 52 |
| 16. Chapter 6 – Process Description..... | 56-58 |
| 17. Chapter 7 – Graphical User Interface Testing..... | 59-68 |
| 18. Chapter 8 – Results and Conclusion..... | 69-70 |
| 19. Contribution of the Project..... | 71 |
| 20. Bibliography..... | 72-75 |
| 21. Appendix-I..... | 76-88 |

LIST OF FIGURES

| Figure No | Figure Caption | Page No. |
|-----------|--|----------|
| 1 | Broad types of processing on data and images. | 1 |
| 2 | Different types of noises with their probability distributions. | 4 |
| 3 | Forward and Inverse Wavelet Transform | 7 |
| 4 | An illustration of the lifting scheme | 8 |
| 5 | A general framework of lifting scheme-based image filters. | 11 |
| 6 | Comparative reconstructed images using Proposed Lifting filter and Adaptive Filter | 16 |
| 7 | Noise cancellation results using lifting filter | 17 |
| 8 | Lena progressive image filtering using Lifting Filter | 17 |
| 9 | Progressive image restoration using our proposed Median based Lifting Filter. | 24 |
| 10 | Results generated by Image Restoration Algorithm for a 512*512 Peppers image. | 24 |
| 11 | Comparative restored images using our proposed Median based Lifting Filter and the ARSMF. | 25 |
| 12 | General Framework of DP-PSO | 32 |
| 13 | Flowchart of our DP-PSO Algorithm | 35 |
| 14 | Comparative reconstructed images using Proposed Neighborhood average filter utilizing DP-PSO and PSO-CNN | 37 |

| | | |
|----|--|----|
| 15 | Noise cancellation results using Proposed Neighborhood average filter utilizing DP-PSO | 37 |
| 16 | DP-PSO Algorithm Simulator | 42 |
| 17 | Comparative reconstructed images using Proposed Neighborhood average filter utilizing DP-PSO and PSO-CNN | 43 |
| 18 | Noise cancellation results on Lena 128*128 image using Proposed Neighborhood average filter utilizing DP-PSO | 44 |
| 19 | Noise cancellation results on Lena 512*512 image using Proposed Neighborhood average filter utilizing DP-PSO | 44 |
| 20 | Two-dimensional CNNs organized in an eight-neighbor rectangular grid. | 46 |
| 21 | Nonlinear Function. | 47 |
| 22 | Functional Model of CNN Architecture | 48 |
| 23 | A CNN Template | 49 |

LIST OF TABLES

| Table No | Table Caption | Page No. |
|----------|--|----------|
| I | COMPARATIVE EVALUATION OF OUR LIFTING FILTER | 16 |
| II | COMPARATIVE EVALUATION OF OUR METHOD (MEDIAN BASED LIFTING FILTER) | 23 |
| III | PERFORMANCE OF OUR METHOD (MEDIAN BASED LIFTING FILTER) | 23 |
| IV | THE INITIALIZED COEFFICIENTS OF THE DP-PSO | 35 |
| V | Comparative Evaluation of our Proposed Neighbourhood Average Filter Optimized by DP-PSO for Image Corrupted by Salt and Pepper Noise (10%) | 36 |
| VI | SIMULATION RESULTS FOR DIFFERENT NOISE DENSITIES FOR OUR PROPOSED OPTIMIZED NEIGHBORHOOD AVERAGE FILTER | 36 |
| VII | The initialized coefficients of the DP-PSO (Improved) | 39 |
| VIII | SIMULATION RESULTS FOR DIFFERENT NOISE DENSITIES FOR OUR PROPOSED OPTIMIZED NEIGHBORHOOD AVERAGE FILTER | 43 |

LIST OF ABBREVIATIONS

| Abbreviation | Full Form |
|--------------|--|
| CNN | Cellular Neural Network |
| DP-PSO | Diminishing Population - Particle Swarm Optimization |
| gBest | global Best |
| GUI | Graphical User Interface |
| GUIDE | Graphical User Interface Development Environment |
| lBest | local Best |
| LFSGW | Lifting Filter using Second Generation Wavelets |
| MSE | Mean Square Error |
| PSNR | Peak Signal to Noise Ratio |
| PSO | Particle Swarm Optimization |

ABSTRACT

The project seeks to develop novel and better methods than the existing ones for Image Processing tasks such as Image Noise Cancellation. The study of existing algorithms, their implementation and development of ways to improve the methods by removing their shortcomings is quintessential to the project.

Towards this end we will make use of two techniques.

The first is the Particle Swarm Optimization (PSO) and the second is the Cellular Neural Networks (CNN).

The main focus of the project is the development of techniques for Image Noise Cancellation which overcome the shortcomings of the existing techniques. The algorithms can be put to effective use by researchers and the industry.

Finally Graphical User Interface (GUI) will be developed to simulate all the developed algorithms so that it is convenient for the end user to grasp the essence of the project.

CHAPTER 1

NOISE IN IMAGE PROCESSING

Vision is a complicated process that requires numerous components of the human eye and brain to work together. The sense of vision has been one of the most vital senses for human survival and evolution. Humans use the visual system to see or acquire visual information, perceive, i.e. process and understand it and then deduce inferences from the perceived information.

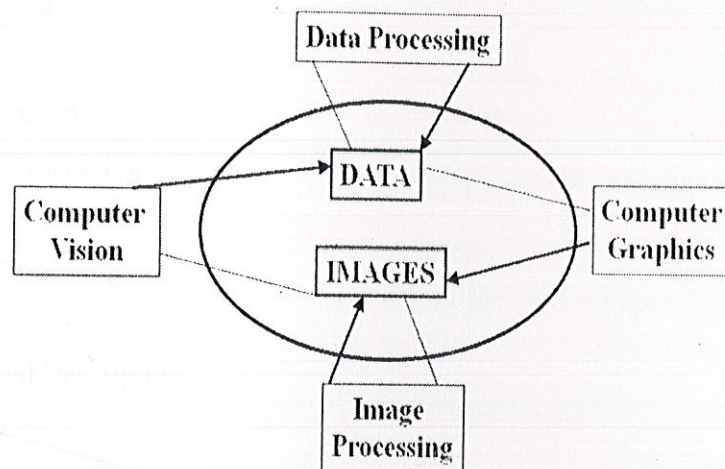


Fig. 1. Broad types of processing on data and images.

The field of image processing focuses on automating the process of gathering and processing visual information. The process of receiving and analyzing visual information by digital computer is called *digital image processing*.

An image may be described as a two-dimensional function I .

$$I = f(x, y)$$

where:

x and y are spatial coordinates.

amplitude of f at any pair of coordinates

(x, y) is called intensity I or gray value of the image.

When spatial coordinates and amplitude values are all finite, discrete quantities, the image is called digital image.

Digital image processing may be classified into various sub branches based on methods whose:

- input and output are images and
- inputs may be images where as outputs are attributes extracted from those images.

Following is the list of different image processing functions based on the above two classes:

- Image Acquisition
- Image Enhancement
- Image Restoration
- Color Image Processing
- Multi-resolution Processing
- Compression
- Morphological Processing
- Segmentation
- Representation and Description
- Object Recognition

For the first seven functions the inputs and outputs are images where as for the rest three the outputs are attributes from the input images. With the exception of image acquisition and display most image processing functions are implemented in software. Image processing is characterized by specific solutions, hence the technique that works well in one area can be inadequate in another. The actual solution of a specific problem still requires a significant research and development.

1.1 Noise

Image noise is a random, usually unwanted, variation in brightness or color information in an image. Image noise can originate in film grain, or in electronic noise in the input device (scanner or digital camera) sensor and circuitry, or in the unavoidable shot noise of an ideal photon detector. Image noise is most apparent in image regions with low signal level, such as shadow regions or underexposed images. High levels of noise are almost always undesirable, but there are cases when lower levels of noise may be useful, for example to prevent discretization

artifacts (color banding or posterization). Noise purposely added for such purposes is called dither. Noise can creep in images during the process of Image Acquisition and Image Transmission due to errors in sensors or in the communication channels. In the project work in Image Noise Cancellation I have considered this common salt and pepper noise.

Noise cannot be removed without the loss of some information in the form of image detail. Nevertheless, noise-reduction algorithms have been developed to reduce noise without degrading image information too much.

1.2 Salt-and-pepper noise

Salt and pepper noise is a type of impulse noise and is seen in images. It represents itself as randomly occurring white and black pixels. Salt and pepper noise either occurs as a very high pixel value or a very low pixel value. Hence it is a type of impulse noise. In the case of a grayscale image the high pixel value is 255 and the low pixel value is 0. A colored image is an extrapolation of a grayscale image and salt and pepper noise when introduced in a color image can be seen corrupting pixels by changing the individual R, G and B component of the pixels to pixel values 0 and 255. This type of noise can be caused by dead pixels, analog-to-digital converter errors, bit errors in transmission, etc. An effective noise reduction method for this type of noise involves the usage of a median filter. Salt and pepper noise creeps into images in situations where quick transients, such as faulty switching takes place.

Fat-tail distributed or "impulsive" noise is sometimes called salt-and-pepper noise or spike noise. An image containing salt-and-pepper noise will have dark pixels in bright regions and bright pixels in dark regions.

1.3 Evaluation Measures

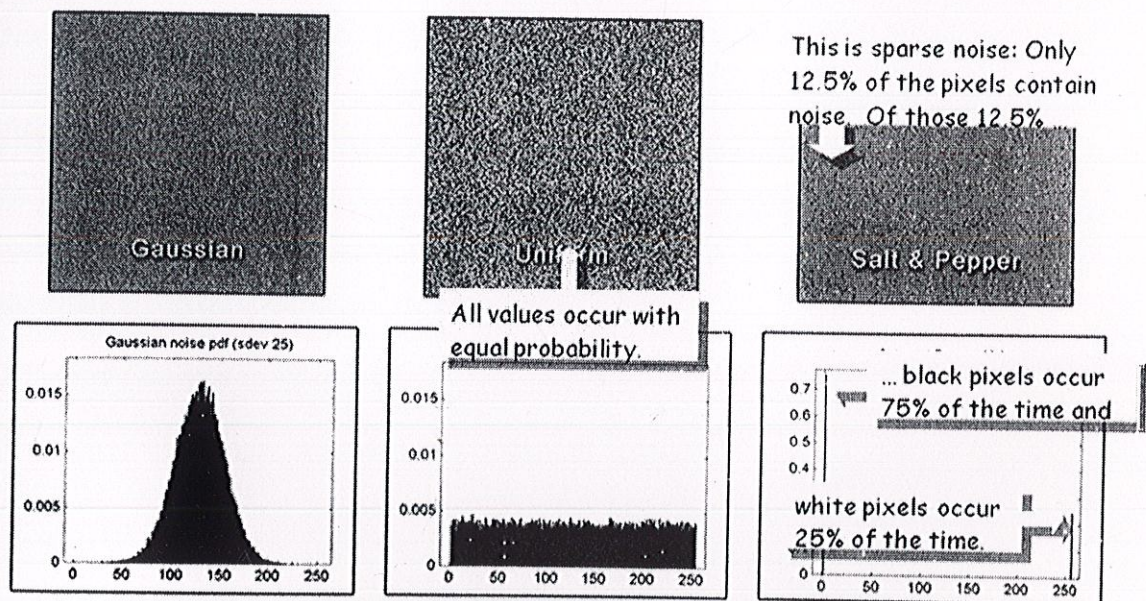
Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) are used to gauge the effectiveness of the algorithms designed to remove noise from images.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \| I(i, j) - R(i, j) \|^2 \quad (1)$$

$$PSNR = 10 \log_{10} \left(\frac{MAX_i^2}{MSE} \right) \quad (2)$$

Where, MAX_i is the maximum possible pixel value of the image. In the case of a grayscale image $MAX_i=255$. 'I' is the original image and 'R' is the reconstructed image using the noise cancellation algorithm. Both the images are of resolution $m \times n$.

MSE gives the amount of noise that is present in the image by finding the sum of the squared differences between the pixel values of the original and reconstructed image. PSNR gives the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. The higher the PSNR the better is the noise removal algorithm. As the signals have a wide dynamic range the PSNR is measured in logarithmic decibel scale.



Each pixel's value has probability of occurrence given by the associated distribution.

Fig. 2. Different types of noises with their probability distributions.

Impulse noise removal consists of detecting the noisy pixel taking into account the edges and substituting the noisy pixel with the best approximation of the correct pixel value based on the neighborhood or detecting the edges, preserve them for blurring and smoothing the locally

smooth and distinct areas. Designing the most complete and sound noise filter is the one of the objectives of our work.

Wavelets have found applications in various fields like geographic modeling, data compression, data manipulation, numerical calculations etc. The simplicity of the methodology to represent data using wavelets is mainly linear and the property of wavelets to represent audio, video, pictures and graphics from a source contains a lot of redundancy and are highly correlated. Wavelet uses this property of the digital data to represent it. It uses very few parameters to represent the digital data precisely.

2.1 Second Generation Wavelets

The Second Generation Wavelets are easier to understand and implement than the First Generation Wavelets. The First Generation Wavelets could be used for periodic and harmonic domain signals. But there was no clear cut way of using it for localized domain signals. Thus the Second Generation Wavelets came into picture. The Second Generation Wavelets have all the useful properties like time frequency localization and fast computation which the First Generation Wavelets have. In addition to being able to represent the signals which are localized. This has been

CHAPTER 2

LIFTING SCHEME

First generation wavelet has become invariable tool in computer graphics due to their ability to compute compact representation of functions and data set. It has been most efficiently applied to generate image accurately and in sampling the data set significantly. The tool that is used in the developed algorithms to build wavelets transforms is known as the second-generation wavelets, which relies on the lifting scheme. The main feature of the lifting scheme is that all functions are derived in the spatial domain. This is in contrast to the initial approach, which relies heavily on the frequency domain. Usage of spatial domain has major advantages that it does not require the machinery of Fourier analysis as a prerequisite, which results in a more intuitively appealing treatment.

Wavelets have found applications in various fields like geometric modeling, data compression, data transmission, numerical computations etc. The complexity of the calculations to represent data using wavelets is normally linear and the process is even fast. Digital data like audio, video, pictures and graphics from a source contains a lot of redundancy and the data is highly correlated. Wavelet uses this property of the digital data to represent it. It uses very few parameters to represent the digital data precisely.

2.1 Second Generation Wavelets

The Second Generation Wavelets are easier to understand and implement than the First Generation Wavelets. The first generation wavelets could be easily used for periodic and infinite domain signals. But there was no clear cut way of using it for bounded domain signals. Thus the Second Generation Wavelets came into picture. The Second Generation Wavelets have all the useful properties like time-frequency localization and fast implementation of the First Generation Wavelets in addition to being able to represent the signals which are bounded. This has been achieved by removing the translation and dilation of the mother wavelet. Instead we use a Lifting Scheme. Hence we do not use any Fourier Analysis.

There are two advantages of second generation wavelets. The first advantage is that fast computation and multiresolution capabilities of the first generation wavelets, are retained in the

second generation wavelets. The second advantage is that the forward and inverse wavelet transform are invertible to each other as can be seen from the figure below.

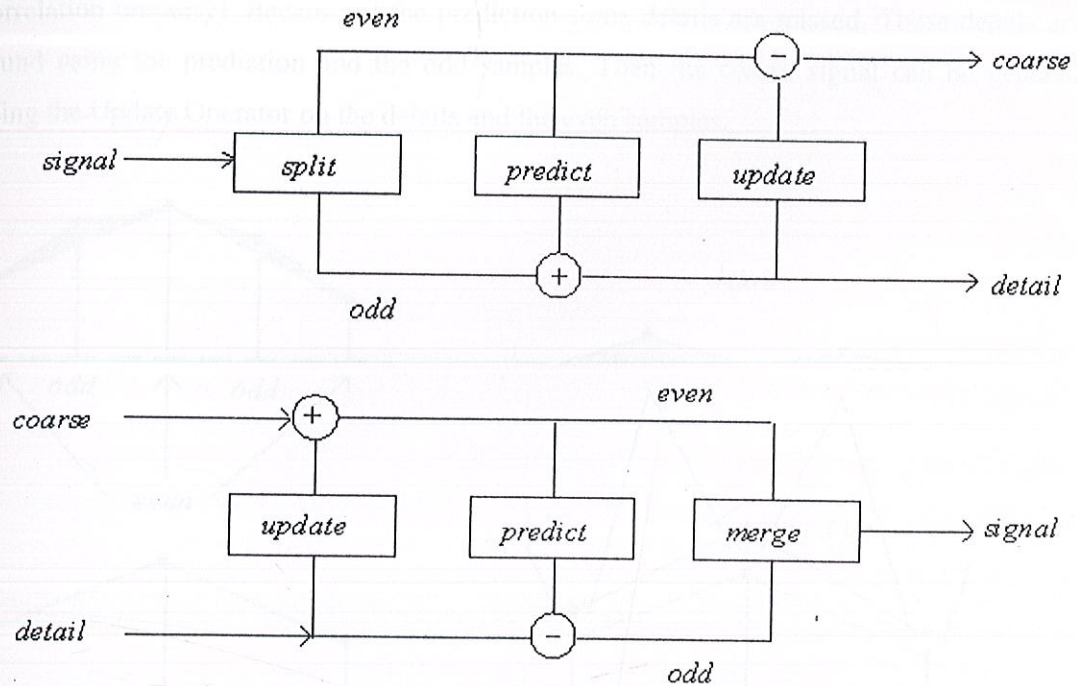


Fig. 3. Forward and Inverse Wavelet Transform

2.2 Lifting Scheme

Lifting scheme is better suited for image de-noising as it can easily be generalized to complex geometric situations of high non-uniformity. The lifting scheme is a tool for constructing second-generation wavelets which are no longer, dilates and translates of one single function the mother wavelet. In contrast to first-generation wavelets, which used the Fourier transform for wavelet construction, a construction using lifting is performed exclusively in spatial domain and, thus, wavelets can be custom designed for complex domains like irregular noise samples.

The lifting scheme can be viewed as a process of taking an existing wavelet and modifying it by adding linear combinations of the scaling function at the same level of resolution.

The scheme consists of three steps: *Split*, *Predict* and *Update*.

In the lifting scheme we use the correlation property of the digital data. We split the

signal by sampling into even and odd samples. We have two operators, the Predict Operator and the Update Operator. Both the operators are linear. We predict the odd samples from the even samples using the Predict Operator (can use prediction as they are correlated because of the correlation property). Because of the prediction some details are missed. These details are then found using the prediction and the odd samples. Then the coarse signal can be generated by using the Update Operator on the details and the even samples.

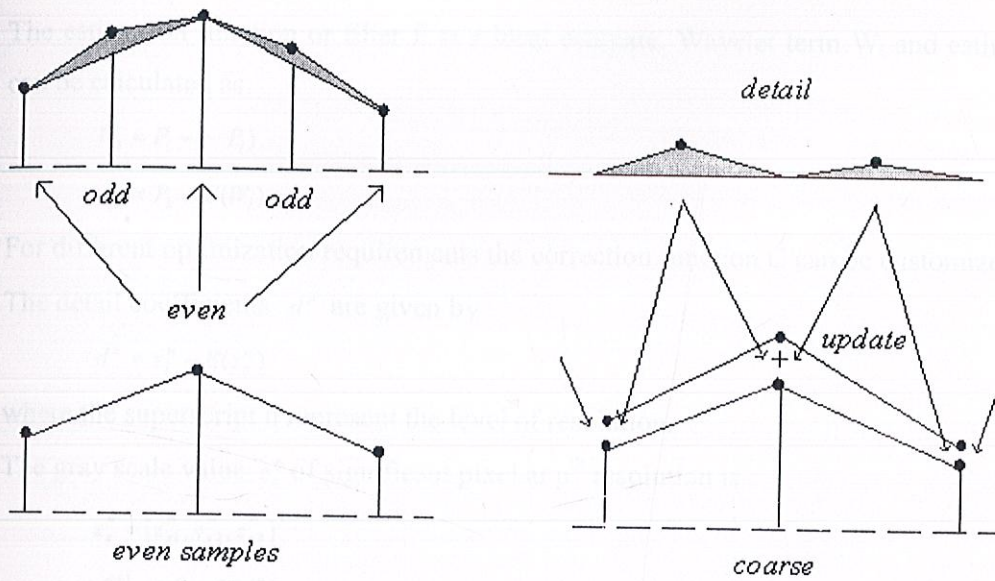


Fig. 4. An illustration of the lifting scheme

i) Split: Consider a data set to be partitioned into two groups: significant points (SP) 'SP' and insignificant pixel (IP) 'IP'. If the original point set can be partitioned in a hierarchical structure, then the above process can be iteratively applied to different sets. A hierarchical structure has the following form

$$SP^0 \subset SP^1 \subset SP^2 \dots SP^n \quad (3)$$

where SP^n denotes the finest representation of the geometry. SP^n can be partitioned into SP^{n-1} and IP^{n-1} ; then SP^{n-1} can be partitioned into SP^{n-2} and IP^{n-2} and so on. Note that the larger number in superscripts represents finer resolution.

$$SP^k \cup IP^k = SP^{k+1}, k = 0, 1, 2, \dots, n-1 \quad (4)$$

Wavelets and approximations of the data can be constructed on the basis of this hierarchical structure of partitioning. In this second-generation wavelet representation, the process does not depend on a regular setting for the data; therefore, it can be used in both the regular and irregular data sets for noise sampling. This is an important advantage of the lifting scheme.

ii) Predict: Using 'P_i' to express position vector of ith point, one can construct filter E to estimate P_i based on P_s

$$\sim P_i = E(P_s) \quad (5)$$

The estimation function or filter E is a local estimate. Wavelet term W_i and estimation term A_s can be calculated as

$$W_i = P_i - (\sim P_i) \quad (6)$$

$$A_s = P_s + C(W_i) \quad (7)$$

For different optimization requirements the correction function C can be customized.

The detail coefficients dⁿ are given by

$$d^n = z_i^n - E(z_s^n) \quad (8)$$

where the superscript n represent the level of resolution.

The gray scale value z_sⁿ of significant pixel at nth resolution is

$$z_s^n = [z_{s1}^n, z_{s2}^n, z_{s3}^n]^T \quad (9)$$

$$z_s^{n+1} = z_s^n + C(d^n) \quad (10)$$

3. Update: The pixels with impulse noise are updated based on the value calculated by the estimation function or filter E. If the results are unsatisfactory then the pixels are included in the significant set from the insignificant set based on the detail coefficient and some threshold value specific to the application. The pixels which are added in the significant data set are simultaneously removed from the insignificant data set. Then, we again update the pixels of significant data set based on the values found by filter E. By iteratively using this process we can get the desired results.

2.3 Application

1. Application 1: Image Noise Cancellation by Lifting Filter using Second Generation Wavelets (LFSGW)

In this application, I have developed a novel Second Generation Wavelets based lifting filter for image noise cancellation. The algorithm finds the most significant impulse noise points

based on the values of their neighboring points, which are then used for noise cancellation. The algorithm works well for grayscale images and it has been compared with the Adaptive Filter with Weight Training Mechanism method. The efficacy of the proposed Lifting Filter method has been shown and the results are gauged using PSNR measures.

Image noise cancellation is a prerequisite step for other image processing like object recognition, edge detection etc. Without noise cancellation the results of such image processing will most likely be flawed. Many algorithms have been developed for image noise cancellation but they have not been put to effective use in the case of noise cancellation using Wavelets. In our algorithm the Second Generation Wavelets based Lifting Scheme has been used for noise cancellation in grayscale images.

I have developed a new lifting filter based on the concept of lifting scheme developed by Siddavatam Rajesh et al. The lifting filter uses the lifting scheme of second generation wavelets. The significant noise samples are lifted based on the values calculated by the lifting filter. The lifting scheme is a tool for constructing second-generation wavelets, which are no longer, translates and dilates of the single mother wavelet function. Median-based filters have the property to retain edges and hence they have been used for impulse noise cancellation in. Median-based filters modify all the pixels of the images leading to good pixels being modified. To overcome this drawback switching scheme based filters were used by researchers, which used impulse detection algorithm for finding impulse noise and then used filters for canceling the noise. This too has a disadvantage that when a noise pixel is surrounded by a number of other noise pixels, the median filter could not give a correct filtered value. Particle Swarm Optimization (PSO) has also been used for image noise cancellation.

The lifting filter has been used for noise cancellation in grayscale images. In each iteration of the algorithm noise is removed from the entire image. In our algorithm the size of the window is increased if the current window size is giving neighboring pixels all of which are having impulse noise. This solves the problem of removing noise blotches from images.

A. Lifting Filter

The lifting filter is used for finding the filtered value for all the pixels which might be noise. Here I have considered homogeneous salt and pepper noise. So if any of the 3 pixel values (r, g, or b) of a pixel has a value of 0 or 255 then there is a possibility that this pixel might be having an impulse noise. Thus the lifting filter is used to find the filtered averaged value of the pixel under consideration based on the value of its neighbors.

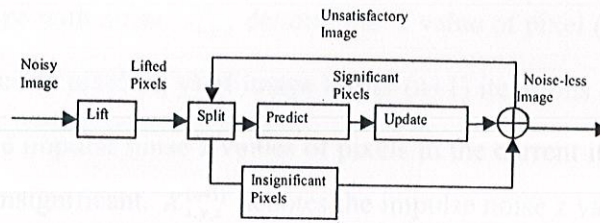


Fig. 5. A general framework of lifting scheme-based image filters.

The notation given below will be followed in this text:

W: Window Size

i and j: the size of the original image $i*j$

$k=3$

x and y: pixel coordinates of pixel (x, y)

z: the r, g or b value of the pixel under consideration

$I_{i,j}$ or $I_{i,j}^0$: Original image of size $i*j$ with noise

$I_{i,j}^{n+1}$: Image after iteration n+1

$X_{x,y,z}^{n+1}$: Array of size $i*j*k$ having impulse z values of pixels after iteration n+1

$\Omega_{x,y,z}^W$: Set of non impulse z values of pixels within a $W*W$ window centered about pixel (x, y)

$sig_{i,j,k}^{n+1}$: set of size $i*j*k$ of significant z values of pixels after iteration n+1

$insig_{i,j,k}^{n+1}$: set of size $i*j*k$ of insignificant z values of pixels after iteration n+1

T: Threshold

$d_{x,y,z}^{n+1}$: Detail of z value of pixel (x, y) after iteration n+1

$a_{x,y,z}^{n+1}$: Filtered value found by the lifting filter of z value of pixel (x, y) after iteration n+1

$$n \in \{0, 1, 2, \dots\}$$

$$W = \{3, 5, 7, \dots\}$$

In the noise cancellation algorithm using Second Generation Wavelets two image sequences are generated as follows:

- 1) The sequence of grayscale images $\{I_{x,y,z}^{(0)}, I_{x,y,z}^{(1)}, I_{x,y,z}^{(2)}, \dots, I_{x,y,z}^{(n)}, I_{x,y,z}^{(n+1)}, \dots\}$.
- 2) The sequence of images having impulse noise $\{X_{x,y,z}^{(0)}, X_{x,y,z}^{(1)}, X_{x,y,z}^{(2)}, \dots, X_{x,y,z}^{(n)}, X_{x,y,z}^{(n+1)}, \dots\}$.

I denotes the input image with noise. $I_{x,y,z}^{(0)}$ denotes the z value of pixel (x, y) in the input image. $I_{x,y,z}^{(n+1)}$ denotes the z value of pixel (x, y) of image I after $(n+1)$ iterations of the noise cancellation algorithm. X has all the impulse noise z values of pixels in the current iteration of the algorithm whether significant or insignificant. $X_{x,y,z}^{(n+1)}$ denotes the impulse noise z value of pixel (x, y) in the $(n+1)^{\text{th}}$ iteration of the algorithm. Since size of X is $i*j*k$ and in almost all cases the image is not entirely filled with noise there will be certain positions of X which will be empty. These empty positions have been given the value of -1 which indicates that the pixel is noise free.

In each iteration of the algorithm we use the lifting filter to find the value ' $a_{x,y,z}$ ' which gives the value which should be given to z (r, g or b) value of pixel (x, y) if the pixel's current z value is giving a detail coefficient which exceeds the threshold value. ' $a_{x,y,z}^{n+1}$ ' gives the value calculated by the lifting filter for z value of pixel (x, y) from its neighboring non impulse pixels in a window of size $W*W$ in the iteration $(n+1)$.

The non impulse pixels in window of size $W*W$ are given by

$$\Omega_{x,y,z}^W = \{j = (j_1, j_2) \mid x - (W-1)/2 \leq j_1 \leq x + (W-1)/2, y - (W-1)/2 \leq j_2 \leq y + (W-1)/2\}. \quad (11)$$

With the help of these pixels we can use the lifting filter to find $a_{x,y,z}^n$ as under

$$a_{x,y,z}^n = \text{lift}\{I_j^n \mid j \in \Omega_{x,y,z}^W\}. \quad (12)$$

The value of $a_{x,y,z}^n$ is used in the predict operation of lifting scheme for finding the details coefficient $d_{x,y,z}^{n+1}$.

B. Noise Cancellation Algorithm:

I. Significant Noise Sampling Algorithm

The following steps are used to obtain a nested subset of significant noise points.

Input: Noisy Image I

- i. Let $I^{n+1} = I : \text{data set } (sig^k \cup insig^k)$
- ii. Using lifting scheme find a set of new significant noise points (SP) to be filtered using the lifting filter.
- iii. Get $sig^k = sig^{k-1} - SP$
- iv. Repeat the step 2 to 3 to get the desired image quality (sig^k).
- v. Check the quality of image obtained from the data. The process is stopped if a good image is generated.

Output: Denoised Image I^{n+1} .

II. Image Noise Cancellation Algorithm

The Noise Cancellation algorithm for grayscale images works by considering all the three r, g and b values of each pixel of the image. In grayscale we know that $r=g=b$, but I've developed the algorithm in a generic way so as to accommodate colored images. Because of the introduction of homogeneous salt and pepper noise the r, g and b values of the pixels may be corrupted. The lifting filter is used to find the filtered value $a_{x,y,z}''$ for the r, g or b value of the pixel under consideration. Then this value is used for finding the detail coefficient. This detail coefficient is then compared with the threshold value. If the detail coefficient exceeds the threshold value the z value of the pixel under consideration is added to the set of significant pixels else it is added to the set of insignificant pixels. Then in the update operation for all the z values in the significant set the pixel value is updated with $a_{x,y,z}''$. Then if the results are not satisfactory we again find a set of significant noise point using the significant noise sampling algorithm and then we use the grayscale image noise cancellation algorithm again. This process is repeated until a good image is obtained. It has been experimentally found that it takes a small number of iterations (usually single iteration) for obtaining a good image. The results of the experiments have been shown in the Results section.

Input: Noisy Image $I_{i,j}$ or $I_{i,j}^0$

1. *Lifting (Split)*: If $I_{x,y,z}^n = 0$ or $I_{x,y,z}^n = 255$ then

$$X_{x,y,z}^n = I_{x,y,z}^n. \quad (13)$$

2. *Predict*:

2.1 The set $X_{x,y,z}^n$ is divided in the sets of significant and insignificant pixels. If $I_{x,y,z}^n = 0$ or $I_{x,y,z}^n = 255$ (salt or pepper) then for the z (r, g or b) whose value is 0 or 255 calculate $a_{x,y,z}^n$ as shown in (14) and (15).

$$\Omega_{x,y,z}^W = \{j = (j_1, j_2) \mid x - (W-1)/2 \leq j_1 \leq x + (W-1)/2, \\ y - (W-1)/2 \leq j_2 \leq y + (W-1)/2\}. \quad (14)$$

$$a_{x,y,z}^n = \text{lift}\{I_j^n \mid j \in \Omega_{x,y,z}^W\} \quad (15)$$

Please note that W can be set to any odd number greater than or equal to 3. In this algorithm W is set to some initial value. For generating the results in this paper we initially set W to 3. Then if while finding $a_{x,y,z}^n$ the z values under consideration of all pixels in the window are themselves noise too (value 0 or 255 in the case of salt and pepper noise), we increase the window size (only for the current pixel; for some other pixel W is the initially set value) to the next higher size (we increased W to 5) and find $a_{x,y,z}^n$ again i.e. step 2.1 is repeated again with new window size. This leads to correct filtered value being used. Also as the window size is kept small until it is essential to increase W , it saves execution time. Moreover impulse noise in groups i.e. noise blotches are easily removed and also the noise is not propagated further in the image.

2.2 Find

$$d_{x,y,z}^{n+1} = |I_{x,y,z}^n - a_{x,y,z}^n|. \quad (16)$$

2.3

$$\text{If}(d_{x,y,z}^{n+1} > T) \\ sig_{x,y,z}^{n+1} = I_{x,y,z}^n \quad (17)$$

$$\text{else} \\ insig_{x,y,z}^{n+1} = I_{x,y,z}^n \quad (18)$$

3. *Update*: For all the significant pixels update

$$I_{x,y,z}^{n+1} = a_{x,y,z}^{n+1}, \text{ if } I_{x,y,z}^{n+1} \in sig_{x,y,z}^{n+1} \quad (19)$$

4. If the results are good stop the process. If the results are not good then find a new set of new significant noise samples. Repeat the steps 2 to 3 and then check the results again.

Output: Denoised Image I^{n+1} .

C. Results:

This application uses the noise cancellation algorithm for denoising images. Here we show the experimental results of noise cancellation carried out using the lena image. We have corrupted the lena image with different levels of noise ranging from 5% to 80%. The reconstructed image in all these cases is visually good compared to the original lena image to which homogeneous salt and pepper noise had been added. The execution time of the algorithm is also found to be very less.

We have used MSE and PSNR to quantitatively evaluate the proposed algorithm for noise cancellation in grayscale images. All images with salt and pepper noise of noise densities less than 10% show excellent PSNR values greater than 44 dB. For images with salt and pepper noise of noise densities between 10% and 20% the PSNR is between 44 dB and 37.56 dB. Even lena image with 50% noise density after noise cancellation gives a PSNR value of 31.8 with a single iteration of the algorithm. The comparative evaluation of our algorithm has been shown in Table I. Noise cancellation for images with noise densities less than 80% has been done with a single iteration of the algorithm. For images having noise densities greater than 80%, two iterations of the proposed algorithm were sufficient to generate a good reconstructed image. Thus even with a small number of runs we are able to generate good images. This saves valuable execution time.

Fig. 6. Comparative reconstructed images using Proposed, Filtering filter and Adaptive filter

TABLE I
COMPARATIVE EVALUATION OF OUR LIFTING FILTER

| Test Image | Algorithm Used | Noise Density | PSNR (in dB) |
|-----------------|-----------------|---------------|--------------|
| Lena 512x512 | Lifting Filter | 10% | 44.1228 |
| | | 20% | 37.5694 |
| | | 50% | 31.8077 |
| | | 80% | 27.2215 |
| | Adaptive Filter | 10% | 40.5690 |
| | | 20% | 32.6834 |
| | | 50% | 26.0444 |
| | | 80% | 20.8967 |

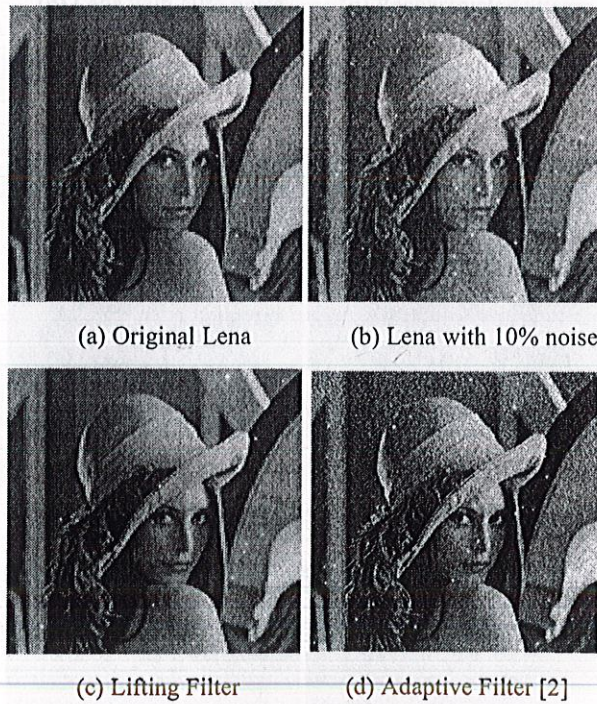
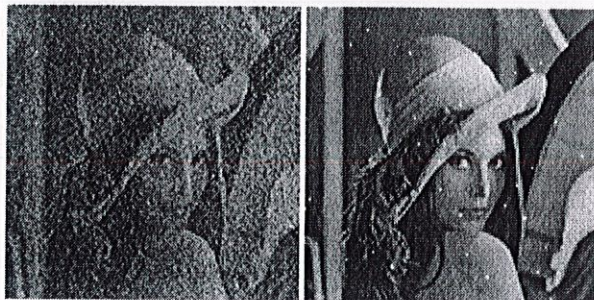


Fig. 6. Comparative reconstructed images using Proposed Lifting filter and Adaptive Filter



(a) Lena with 20% noise

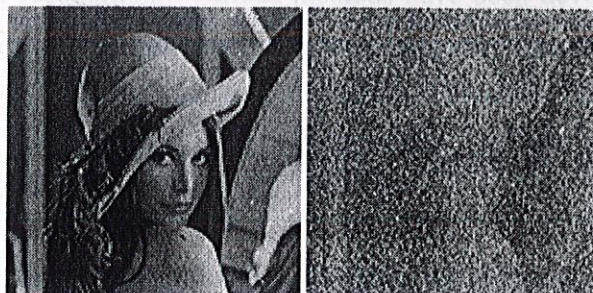
(b) Reconstructed image



(c) Lena with 50% noise

(d) Reconstructed image

Fig. 7. Noise cancellation results using lifting filter



(a) Original Lena

(b) Lena with 80% noise



(c) Lifting Scheme – run1

(d) Lifting Scheme – run2

Fig. 8. Lena progressive image filtering using Lifting Filter

2. Application 2: Fast Image Restoration by Median Based Lifting Filter using Second Generation Wavelets

In this application we have developed a novel median based Lifting Filter for restoring images which have been corrupted by homogeneous salt and pepper noise. The median based lifting filter restores the input image by calculating the median of the neighboring significant pixels. The algorithm for image restoration uses the lifting scheme of the second generation wavelets in conjunction with the median based Lifting filter. The experimental results demonstrate the efficiency of the proposed method. The algorithm works exceedingly well for all levels of noise, as is illustrated by the Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) measures.

In the process of image acquisition and transmission the digital images are often corrupted with impulse noise. This is mainly because of the errors in the sensors or in the communication channel. It is highly imperative that this noise be removed and the digital image restored prior to its subjection to further processing, such as edge detection, image segmentation, object recognition and pattern recognition. Various methods have been developed for impulse noise removal from corrupted images. Median filters have been used extensively for the removal of impulse noise. They are simple yet very effective in the removal of salt and pepper type impulse noise. Median filters often tend to modify good pixels too. Therefore impulse detection algorithms play a crucial role in noise removal.

The Progressive Switching Median (PSM) filter has been developed for removing impulse noise from highly corrupted images. It works by using an impulse detection algorithm and then iteratively detecting and filtering impulse noise and hence it performs heavy computation.

An Adaptive Rank-ordered Switching Median Filter has been used by researchers for noise removal from images corrupted by salt and pepper noise. It detects corrupted pixels even in case of images which are highly corrupted. Our empirical results have been compared with the ARSMF to show the superiority of our proposed algorithm utilizing median based lifting filter.

The general scheme followed for noise removal using first generation wavelets is described as under. Consider an original image denoted as f and noisy image denoted as Z . Here we assume

that the original image f is corrupted by homogeneous salt and pepper noise resulting in the image Z . Hence we get a model of the type

$$Z(i, j) = f(i, j) + \varepsilon_{i,j} \quad (20)$$

where ε is homogeneous salt and pepper noise.

In our application we have developed a novel median based Lifting Filter which uses the second generation wavelets. We use the lifting scheme firstly for separating the significant pixels from the insignificant pixels and then for filtering the corrupted image using the median based lifting filter. The lifting filter was used in Application 1 by us for noise removal. Upon improving the lifting filter in Application 1 by calculating the median value of the neighboring pixels of the corrupted pixel in its window we were able to improve the results previously generated. The proposed algorithm gives excellent results for all levels of noise.

Second generation wavelets developed by Swelden have been efficiently used for many applications of image processing. Generating set of most significant samples for de-noising and then using them to generate an image is a highly non-linear and computationally expensive task. A set of significant de-noising samples is obtained after estimating the detail coefficients. Highly sparse noise removed significant samples are used to approximate an image.

A. Median Based Lifting Filter:

The median based lifting filter is used for finding the correct filtered value by which a corrupted noisy pixel should be replaced. For finding the filtered value for a noisy pixel under consideration the median based lifting filter considers a window around the noisy pixel and finds the filtered median value of the non impulse noise pixels (noise free pixels) which are neighbors of the corrupted pixel and are present in that window. In case there is no non impulse neighbor of the corrupted pixel in the current window the filter increases the window size to the next possible higher value and calculates the filtered median value. The initial separation of impulse noise pixels from the other image pixels is done in the Lifting Step of the Image Restoration Algorithm. This separation is done to enable the median based lifting filter to calculate the filtered value in the Predict step. After the calculation of the filtered value the final separation of impulse noise pixels and image pixels is done and the significant and insignificant sets are created. The Lift and Predict steps are explained in Section IV. The median based lifting filter always uses a

window size of 3*3 initially to determine the filtered value for a corrupted pixel. As mentioned above if there is no non impulse neighbor of the corrupted pixel in this window of size 3*3, the window size is then increased to the next higher odd integer and the window size becomes 5*5. In this way the filter dynamically increases the window size as and when needed. After this when the filter considers some other noisy pixel the initial window size is again 3*3.

The non impulse neighbors of the corrupted pixel in a window of size W*W are given by

$$\Omega_{x,y}^W = \{j = (j_1, j_2) \mid x - (W-1)/2 \leq j_1 \leq x + (W-1)/2, \\ y - (W-1)/2 \leq j_2 \leq y + (W-1)/2\}. \quad (21)$$

Where (x, y) are the coordinates of the corrupted pixel. Here W can be any odd integer greater than or equal to 3. But while calculating the filtered value for a new corrupted pixel its value is 3 and can be increased later as explained above.

After finding the non impulse neighbors above the median based lifting filter can then calculate the filtered median value $med_{x,y}^n$ as given below

$$med_{x,y}^n = med_lift\{I_j^n \mid j \in \Omega_{x,y}^W\}. \quad (22)$$

Where n is the current number of iteration of the image restoration algorithm.

B. Image Restoration Algorithm:

The Image Restoration Algorithm works by following the three steps of the lifting scheme Split, Predict and Update. In the first step we lift (split) all the pixels of the corrupted input image that are suspected to be noisy pixels. In the case of salt and pepper noise this is done by separating those pixels whose grayscale image value (z) is either 0 or 255 because these are the grayscale values which correspond to pepper and salt. Images corrupted by salt and pepper noise are contaminated with pixels which have either a very high value (255 in grayscale image) or a very low value (0 in a grayscale image). For the current iteration n of the algorithm these pixels are stored in an array $X_{x,y,z}^{n+1}$. Now some of the pixels in the array $X_{x,y,z}^{n+1}$ can also be image data. To separate those pixels we perform the Predict step of our algorithm. In this step we firstly find $med_{x,y}^n$ using the median based lifting filter. Then we calculate the detail using (15); We then compare the detail of the pixel (x, y) under consideration with threshold T to determine whether this pixel (x, y) is corrupted or not. As the data of images is smooth varying the neighboring

pixels of any pixel will not have values which are significantly different from its value. Hence the filtered value found by the median based lifting filter will also be close to the value of the pixel (x, y) if it is image data and is not a noisy pixel. This is the reason why upon comparing the detail with a suitably narrow threshold we can successfully separate image data from noisy data. In our work we put the noisy pixels and image pixels in iteration n in $sig_{x,y}^n$ and $insig_{x,y}^n$, where (x, y) are the coordinates of the pixel. Lastly, in the update step we correct all the pixels in $sig_{x,y}^{n+1}$ with their filtered median value, where $(n+1)$ is the current number of iteration for which the algorithm has been executed. After executing the three steps above if the result is unsatisfactory, we execute the algorithm again on the output of the previous run.

Input: Noisy Image $I_{i,j}$ or $I_{i,j}^0$

1. *Lifting (Split)*: If $I_{x,y}^n = 0$ or $I_{x,y}^n = 255$ then

$$X_{x,y}^n = I_{x,y}^n. \quad (23)$$

2. *Predict*: If $I_{x,y}^n = 0$ or $I_{x,y}^n = 255$

$$\Omega_{x,y}^W = \{j = (j_1, j_2) \mid x - (W-1)/2 \leq j_1 \leq x + (W-1)/2, \\ y - (W-1)/2 \leq j_2 \leq y + (W-1)/2\}. \quad (24)$$

$$med_{x,y}^n = med_lift\{I_j^n \mid j \in \Omega_{x,y}^W\}. \quad (25)$$

Find Detail

$$d_{x,y}^{n+1} = |I_{x,y}^n - med_{x,y}^n|. \quad (26)$$

Build Significant and Insignificant sets

$$\text{If } (d_{x,y}^{n+1} > T) \quad (27)$$

$$sig_{x,y}^{n+1} = I_{x,y}^n$$

else

$$insig_{x,y}^{n+1} = I_{x,y}^n \quad (28)$$

3. *Update*: For all the significant pixels update

$$I_{x,y}^{n+1} = med_{x,y}^n, \text{ if } I_{x,y}^{n+1} \in sig_{x,y}^{n+1} \quad (29)$$

4. Repeat steps 1 to 3 on the output of the previous run until satisfactory results are obtained.

Output: Restored Image I^{n+1} .



C. Results:

We have tested our proposed algorithm for different levels of noise ranging from as low as 5% to as high as 95%. We have introduced homogeneous salt and peppers noise in the images.

The experimental results have been gauged using the Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) measures which have been given below.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I(i, j) - R(i, j))^2 \quad (30)$$

where I and R are the original and the restored images having a resolution of m*n.

$$PSNR = 10 \log_{10} \left(\frac{\max^2}{MSE} \right) \quad (31)$$

where max is the maximum possible pixel value of the image and its value is 255 in the case of a grayscale image.

Our algorithm utilizing median based lifting filter generates results that are superior to the existing methods for noise removal. In this text we compare our results with the Adaptive Rank-ordered Switching Median Filter. The comparative evaluation of our proposed method with ARSMF has been done in Table II below.

Our algorithm gives excellent PSNR values greater than 43 dB for noise ratios less than or equal to 10%. For noise ratios between 10% and 20% the PSNR varies between 43.0821 dB and 39.0591 dB. Even for a noise ratio of 50% we get a PSNR of 33.2258 dB with 2 iterations of the algorithm. All the results generated for noise ratios less than 50% have been generated by a single iteration of the algorithm. For noise ratios of 80% and 95% we get PSNR values of 28.4037 dB and 23.8363 dB respectively. It took 3 iterations of the algorithm to get the results in the case of 80% noise and 4 iterations of the algorithm in the case of 95% noise.

The proposed algorithm has a complexity of $O(n^2)$. For implementing our algorithm we have used MATLAB on a 1.73GHz Pentium M Processor with 256 MB of RAM. The algorithm has been found to be pretty fast. It takes 10.0469 seconds for restoring a 512*512 Lena image corrupted with 5% noise. This is done in a single iteration of our algorithm. For restoring 512*512 Lena image corrupted with 95% noise we needed 4 iterations of our algorithm and the execution took 96.3907 seconds. The execution time of our algorithm together with PSNR for

different levels of noise has been shown in Table III. Hence the algorithm can be seen to be efficient both in terms of its dexterity to restore a corrupted image exceedingly well and to be less computation intensive.

TABLE II
COMPARATIVE EVALUATION OF OUR METHOD

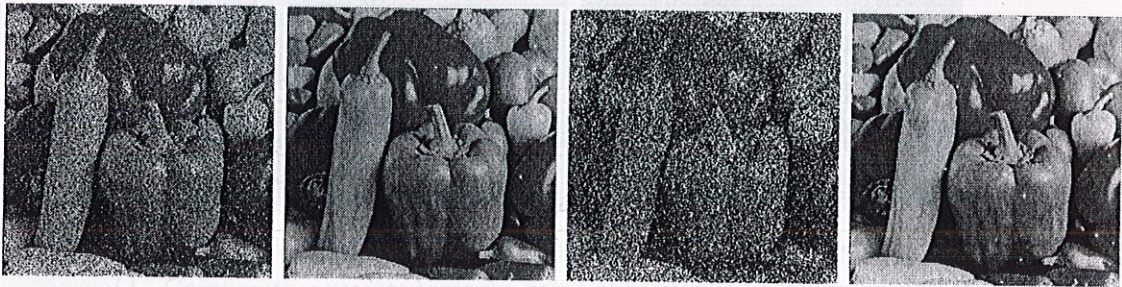
| Test Image | Algorithm Used | Noise Density | PSNR (in dB) |
|-----------------|----------------|---------------|--------------|
| Lena 512*512 | Our Method | 10% | 43.0821 |
| | | 20% | 39.0591 |
| | | 50% | 33.2258 |
| | | 80% | 28.4037 |
| | ARSMF | 10% | 40.8934 |
| | | 20% | 37.1464 |
| | | 50% | 33.4131 |
| | | 80% | 26.5735 |

TABLE III
PERFORMANCE OF OUR METHOD

| Test Image | Algorithm Used | Noise Density | PSNR (in dB) | Execution Time (in sec) |
|-----------------|----------------|---------------|--------------|-------------------------|
| Lena 512*512 | Our Method | 5% | 46.1531 | 10.0469 |
| | | 10% | 43.0821 | 12.9375 |
| | | 20% | 39.0591 | 16.5469 |
| | | 50% | 33.2258 | 34.5469 |
| | | 80% | 28.4037 | 55.1875 |
| | | 95% | 23.8363 | 96.3907 |



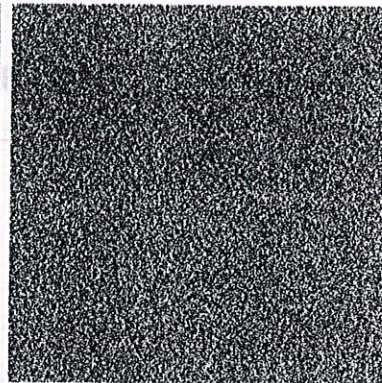
(a) Lena with 80% noise (b) Algorithm: Run 1 (c) Algorithm: Run 2 (d) Algorithm: Run 3
 Fig. 9. Progressive image restoration using our proposed Median based Lifting Filter.



(a) Peppers with 20% noise (b) PSNR=36.7249 dB (c) Peppers with 50% noise (d) PSNR=30.8814 dB
 Fig. 10. Results generated by Image Restoration Algorithm for a 512*512 Peppers image.



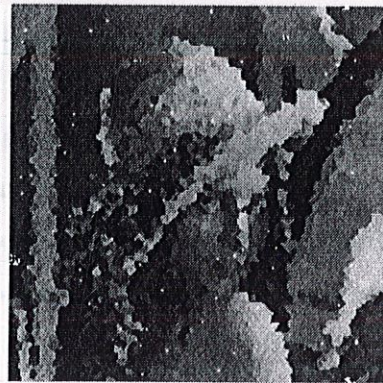
(a) Original Lena



(b) Lena with 95% noise



(c) Proposed Median Based Lifting Filter



(d) ARSMF [1]

Fig. 11. Comparative restored images using our proposed Median based Lifting Filter and the ARSMF.

CHAPTER 3

DIMINISHING POPULATION PARTICLE SWARM OPTIMIZATION

3.1 Particle Swarm Optimization

The particle swarm optimization, first proposed by Kennedy and Eberhart, is a population-based optimization method and can be easily implemented to solve a wide array of different optimization problems. Some example applications include neural network training and function minimization.

The origins of PSO are sociologically inspired since the original algorithm was based on the observed sociological behavior associated with bird flocking. To understand the behavior let us suppose the following scenario: a group of birds are randomly searching for food in an area and only one piece of food is present in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So what could be the best strategy to find the food? One effective strategy is to follow the bird that is nearest to the food. In PSO, each single solution is synonymous to a "bird" in the search space and is referred to as a "particle". All of particles have their own fitness values which are then evaluated by a fitness function which in turn is to be optimized, and all particles also have velocities which direct the flying of the particles. The particles move through the problem space by following the current optimum particles.

The algorithm maintains a population of particles, where each particle i represents a potential solution to an optimization problem and can be represented as an object with several characteristics. Let 's' be the size of the swarm. The characteristics are assigned the following symbolic representation:

x_i : The current position of the particle

v_i : The current velocity of the particle

y_i : The personal best position of the particle

The personal best position associated with particle i is the best position that the particle has visited so far (a previous value of x_i), giving the highest fitness value for that particle. If our

objective is a minimization task then a position yielding a smaller function value will have a higher fitness value. The symbol 'f' is used to denote the objective function being minimized.

The update equation for the personal best position is presented in the following equation, with the dependence on the time t clearly expressed.

$$y_i(t+1) = \begin{cases} y_i(t) & \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{if } f(x_i(t+1)) < f(y_i(t)) \end{cases} \quad (32)$$

Major steps in PSO:

- 1) Initialize a group of random particles (solutions)
- 2) Search for optima by updating generations.

With every iteration, each particle is updated by following two "best" values. The first value is the one mentioned above i.e. the best solution (fitness) it has achieved so far called pbest. Another "best" value tracked by the particle swarm optimizer is the best value obtained so far by any particle in the population which serves as a global best and is called gbest. If a particle takes only a part of the population as its topological neighbors then the best value becomes a local best and is called lbest.

After determining the two best values, the particle updates its velocity with following equation (33).

$$v_{ij}(t+1) = v_{ij}(t) + c_1 * r_1(t) * [y_{ij}(t) - x_{ij}(t)] + c_2 * r_2(t) * [gbest(t) - x_{ij}(t)] \quad (33)$$

The position of each particle is updated using the newly computed velocity vector for that particle, so that

$$X_i(t+1) = x_i(t) + v_i(t+1) \quad (34)$$

The algorithm makes use of two independent random sequences, $r_1 \sim U(0,1)$ and $r_2 \sim U(0,1)$ which are used to affect the stochastic nature of the algorithm as shown in equation (33). The values of r_1 and r_2 are scaled by constants $0 < c_1, c_2 \leq 2$, called the acceleration coefficients and their objective is to influence the maximum size of the step taken by a particle in a single iteration. The velocity update step has been specified separately for each dimension $j \in 1 \dots n$, so that v_{ij} denotes the j^{th} dimension of the velocity vector associated with the i^{th} particle.

From the velocity update equation it can be observed that c_2 regulates the maximum step size in the direction of the global best particle, and c_1 regulates the step size in the direction of the personal best position of that particle. The value of v_{ij} is clamped to the range $[-v_{\max}, v_{\max}]$ to reduce the probability that the particle might leave the search space. If the search space is defined by the bounds $[-x_{\max}, x_{\max}]$, then the value of v_{\max} is typically set so that $v_{\max} = k * x_{\max}$, where $0.1 \leq k \leq 1.0$.

The **pseudo code** for the algorithm is as follows:

For each particle

 Initialize particle population and PSO parameters

End

Do

 For each particle

 Calculate fitness value

 If the fitness value is better than the best fitness value (pBest) in history

 set current value as the new pBest

 End

 Choose the particle with the best fitness value of all the particles as the gBest

 For each particle

 Calculate particle velocity according equation (33)

 Update particle position according equation (34)

 End

While maximum iterations or minimum error criteria is not attained

A brief description of how the algorithm works is as follows: Initially, based on its fitness value some particle is identified as the best particle in a neighborhood of particles. All the other particles are then accelerated in the direction of this particle, and also in the direction of their own best solutions that they might have discovered previously. At times the particles might overshoot their targets i.e. start exploring the search space beyond the current best particles. Additionally all particles have the opportunity to discover better particles en route, in which case the other particles will change direction and head towards the newly discovered 'best' particle. Since most functions have some continuity, chances are that a good solution will be surrounded

by equally good, or better, solutions. By approaching the current best solution from different directions in search space, chances are good that these neighboring solutions will be discovered by some of the particles.

Equation (33) shows that the velocity term v_i of a particle is influenced by three parts, the “momentum”, the “cognitive”, and the “social” part. The “momentum” term $v_{ij}(t)$ represents the previous velocity term which is used to carry the particle in the direction it has travelled so far; the “cognitive” part, $c_1 * r_1(t) * [y_{ij}(t) - x_{ij}(t)]$, represents the tendency of the particle to return to the best position it has visited so far; the “social” part, $c_2 * r_2(t) * [gbest(t) - x_{ij}(t)]$, represents the tendency of the particle to be attracted towards the position of the best position found by the entire swarm.

Position $gbest(t)$ in the “social” part is the best position found by particles in the neighborhood of the i th particle. Different neighborhood topologies can be used to control information propagation between particles. Examples of neighborhood topologies include ring, star, and von Neumann. Constricted information propagation as a result of using small neighborhood topologies such as von Neumann has been shown to perform better on complex problems, whereas larger neighborhoods generally perform better on simpler problems. Generally speaking, a PSO implementation that chooses $gbest(t)$ from within a restricted local neighborhood is referred to as *lbest* PSO, whereas choosing $gbest(t)$ without any restriction (hence from the entire swarm) results in a *gbest* PSO.

Inertia Weight

Observe that the positions $y_{ij}(t)$ and $gbest(t)$ in Eq. (33) can be collapsed into a single term p without losing any information:

$$v_{ij}(t+1) = v_{ij}(t) + c * r(t) * [p - x_{ij}(t)] \quad (35)$$

$$\text{where } p = \frac{c_1 * r_1(t) * y_{ij}(t) + c_2 * r_2(t) * gbest(t)}{c_1 * r_1(t) + c_2 * r_2(t)}$$

$$\text{and } c * r(t) = c_1 * r_1(t) + c_2 * r_2(t).$$

Note that p represents the weighted average of the $y_{ij}(t)$ and $gbest(t)$. It can be seen that the previous velocity term in Eq. (35) tends to keep the particle moving in the current direction. A

coefficient *inertia weight*, w , can be used to control this influence on the new velocity. The velocity update (33) can be now revised as:

$$v_{ij}(t+1) = w * v_{ij}(t) + c_1 * r_1(t) * [y_{ij}(t) - x_{ij}(t)] + c_2 * r_2(t) * [g_{best}(t) - x_{ij}(t)] \quad (36)$$

The inertia-weighted PSO can converge under certain conditions even without using V_{max} . For $w > 1$, velocities increase over time, causing particles to diverge eventually beyond the boundaries of the search space. For $w < 0$, velocities decrease over time, eventually reaching 0, resulting in convergence behavior.

PSO parameter control

One of the advantages of PSO includes the fact that PSO takes real numbers as particles. For example, if we try to compute the solution for $f(x) = x_1^2 + x_2^2 + x_3^2$, the particles can be set as (x_1, x_2, x_3) , and fitness function can be taken as $f(x)$. Then standard procedure can be used to find the optimum value. Search is a repeat process, and the stopping condition can be reaching the maximum iteration number or satisfaction of the minimum error condition.

There are not many parameter that need to be tuned in PSO. Here is a list of the parameters and their typical values.

The number of particles: the typical range is 20 - 40. Actually for most of the problems 10 particles is large enough to get good results. For some difficult or special problems, one can try 100 or 200 particles as well.

Dimension of particles: Determined by the problem to be optimized,

Range of particles: Determined by the problem to be optimized.

Vmax: it determines the maximum change one particle can acquire during one iteration. Usually the range of the particle is set as the Vmax for example, the particle (x_1, x_2, x_3) x_1 belongs $[-10, 10]$, then $V_{max} = 20$.

Learning factors: c_1 and c_2 usually equal to 2. Generally c_1 equals to c_2 and ranges from $[0, 4]$.

The stop condition: the maximum number of iterations the PSO must execute or the minimum error requirement is satisfied. This stop condition depends on the problem to be optimized.

Global version vs. local version: Two versions of PSO exist- Global and Local version.

Gbest Model

- Offers a faster rate of convergence at the cost of robustness.
- Maintains only a single “best solution” called the global best particle, across all the particles in the swarm which acts as an attractor, pulling all the particles towards it.
- Eventually all particles converge to this position. The drawback is that if it is not updated regularly it might lead to premature convergence of swarm.
- The update equation for $gbest = \hat{y}$ and v_i are presented below.

$$\begin{aligned}\hat{y}(t) &\in \{y_0(t), y_1(t), \dots, y_s(t)\} | f(\hat{y}(t)) = \min\{f(y_0(t)), f(y_1(t)), \dots, f(y_s(t))\} \\ v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_j(t) - x_{i,j}(t)]\end{aligned}\quad (37)$$

Note that \hat{y} is called the global best position, and belongs to the particle referred to as the global best particle.

Lbest model

- Tries to prevent premature convergence by maintaining multiple attractors.
- A subset of particles is defined for each particle from which the local best particle, \hat{y}_i , is then selected.
- The symbol \hat{y}_i is referred to as the local best position, or the neighborhood best.
- Assuming that the particle indices wrap around at s , the lbest update equations for a neighborhood of size l are follows:

$$\begin{aligned}N_i &= \{y_{i-l}(t), y_{i-l+1}(t), \dots, y_{i-1}(t), y_i(t), y_{i+1}(t), \dots, y_{i+l}(t)\} \\ \hat{y}_i(t+1) &\in N_i | f(\hat{y}_i(t+1)) = \min\{f(a)\}, \forall a \in N_i \\ v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_i(t) - x_{i,j}(t)]\end{aligned}\quad (38)$$

It can be observed that the particles selected to be in the subset N_i have no relationship with each other in the search space domain i.e. selection is based purely on the particle's index number. This is done for two main reasons: it is computationally cheap, since no clustering has to be performed, and it helps promote the spread of information regarding good solutions to all

particles, regardless of their current location in search space. This model converges somewhat more slowly than the gbest version, but it is less likely to become trapped in an inferior local minimum. One can use global version to get quick result and use local version to refine the search.

3.2 Diminishing Population PSO

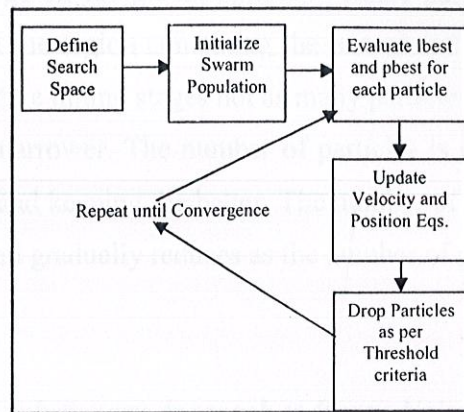


Fig. 12. General Framework of DP-PSO

Particle Swarm Optimization (PSO) is a novel heuristic search methodology for the systematic exploration of solution spaces existing in complex optimization problems. Unfortunately this technique suffers from relatively long execution times as thousands times of iterations require the many repetitions of the update step in order to converge the swarm on the global optimum. Hence other possibilities for dynamic population size improvements for classical PSO were explored by us with the aim of reducing execution time. We came across two techniques to speed up PSO execution. First was the Expanding Population PSO (EP-PSO) which begins with a small number of particles and iteratively increases the swarm size. The other one was Diminishing Population PSO (DP-PSO) which starts with a large number of particles and iteratively reduces the swarm size. Research simulations demonstrate that both improvements bring about almost 60% reduction in the execution time as compared to the classical PSO. However, the results also establish that EP-PSO fares quite badly where the ability to converge to the global optimum is concerned. On the other hand DP-PSO performs reasonably compared to the classical PSO but at much faster convergence and execution speeds. Clearly, DP-PSO

exhibited a lot of promise as an enhancement tool for the classical PSO. In this technique we had to start with a large number of particles and use a pruning process to kill off particles wandering in regions of the space with less promising results. A large population size is favored in the beginning of DP-PSO concept to allow exploration of the widest possible region of the solution space in order that the region containing the global optimum can be recognized. Once that region is identified, the process simply fine tunes the particle positions to converge on the global optimum. Many particles are required in the early initial phases to ensure that the region containing the true global optimum is recognized by the algorithm. During the later fine tuning stages not as many particles are necessary since the search space now becomes much narrower. The number of particles is gradually reduced by dropping lower performing particles and keeping the better. The number of calculations and updates in the early stages remains high and gradually reduces as the number of particles is reduced.

3.3 Application

1. Application 1: An Evolutionary Approach to Image Noise Cancellation Utilizing Diminishing Population Particle Swarm Optimization (DP-PSO)

In this application we have devised a novel method which is an effective implementation of Diminishing Population Particle Swarm Optimization aiming at optimizing the noise removal process in the case of grayscale images contaminated with salt and pepper noise. A new neighborhood average filter has been used in conjunction with DP-PSO for noise removal. Simulations reveal that the proposed scheme which has been designed specifically for noise removal works well in suppressing noise impulses in images corrupted with different levels of noise. The results of the algorithm are compared with those obtained by PSO-CNN method for gray-scale image noise cancellation.

Since the last decade optimization techniques inspired by swarm intelligence and characterized by a decentralized way of working that mimics the behavior of swarms of social insects, flocks of birds, or schools of fish have become increasingly popular. During the past several years, PSO algorithms and its variants have been successfully applied to optimization problems ranging from classical problems such as the travelling salesman problem to highly specialized applications

such as biomedical image registration, parametric object recognition, optimization of templates of cellular neural network used to diminish the noise interference in binary images.

We have focused on a noise removal methodology characterized by the detection of noisy pixels and their subsequent removal using a novel neighborhood average filter which has been optimized by utilizing DP-PSO.

Each DP-PSO iteration accelerates the current pixel value towards the neighborhood average value. Here the distance measure from the neighborhood average value serves as the objective function to be optimized for better performance by our proposed DP-PSO algorithm for image noise cancellation.

Algorithm for Image Noise Cancellation Using DP-PSO

The neighborhood average filter finds the average value of all the suspected non impulse pixels (pixels with values 0 or 255) in a neighborhood of $m \times m$ (where m is an integer greater than 2).

For generating the results shown we have set m to 3 which yields PSNR values which are better than the PSNR values generated by setting m to any other higher value.

Each noisy pixel is considered as a particle constituting the swarm population and is given an initial velocity and position. These velocity and position attributes are then updated according to the DP-PSO algorithm and hence each noisy pixel accelerates to converge towards the average pixel value of the neighborhood in each iteration.

The distance measure from the neighborhood average value serves as the objective function. When the pixel particle attains the $lBest$ value of the neighborhood it is subsequently removed from the swarm. The neighborhood average filter has been used iteratively through the implementation of DP-PSO for removing the noise from the corrupted image. Hence the neighborhood average filter has been optimized by DP-PSO. We first initialize the coefficients of the DP-PSO approach. The Table below gives the initialized coefficients of DP-PSO.

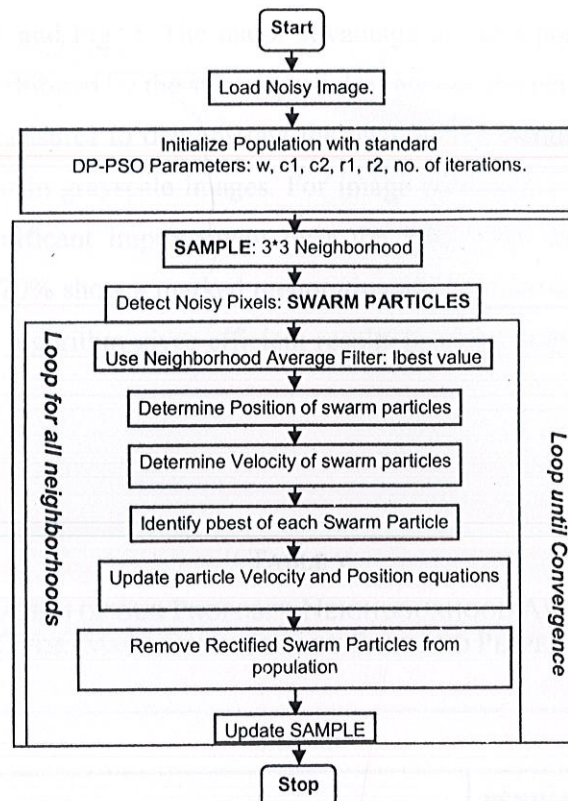


Fig. 13. Flowchart of our DP-PSO Algorithm

TABLE IV
THE INITIALIZED COEFFICIENTS OF THE DP-PSO

| Coefficients | Value |
|----------------------|-------|
| w | 0.4 |
| Number of iterations | 2 |
| c1 | 1 |
| c2 | 1 |

Results

Our neighborhood average filter algorithm has been significantly optimized through the use of DP-PSO. In this section the simulation results carried out for the Lena image are shown. Here the Lena images are corrupted with salt and pepper noise of different levels ranging from 5% to 20%. The reconstructed images show significant improvement over their noisy counterparts as

can be seen from Fig. 1 and Fig. 2. The major advantage of our algorithm can be gauged from the rapid convergence exhibited by the swarm particles towards the neighborhood average value. We have used PSNR measures to demonstrate the quantitative evaluation of our algorithm for image noise cancellation in grayscale images. For image corrupted with 10% noise density our algorithm shows a significant improvement over the PSO-CNN methodology. Also images corrupted with 5% and 20% show a marked melioration after reconstruction. Hence performance analysis shows that our algorithm gives efficient results in terms of evaluation time, complexity and noise cancellation.

TABLE V
COMPARATIVE EVALUATION OF OUR PROPOSED NEIGHBOURHOOD AVERAGE FILTER OPTIMIZED BY DP-PSO FOR IMAGE CORRUPTED BY SALT AND PEPPER NOISE (10%)

| Lena.tif | |
|--|----------|
| | PSNR(dB) |
| PSO-CNN [1] | 21.4748 |
| Proposed Optimized Neighborhood average Filter | 31.9214 |

TABLE VI
SIMULATION RESULTS FOR DIFFERENT NOISE DENSITIES FOR OUR PROPOSED OPTIMIZED NEIGHBORHOOD AVERAGE FILTER

| Test Image | Noise Density | PSNR (in dB) |
|-----------------|---------------|--------------|
| Lena 128x128 | 5% | 35.2005 |
| | 10% | 31.9214 |
| | 20% | 29.2432 |
| | 50% | 275.8293 |

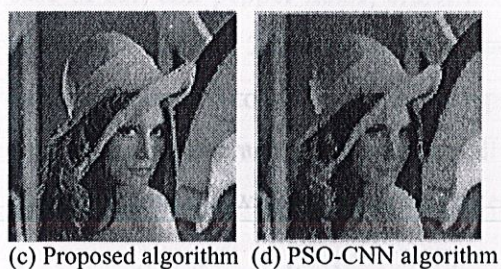
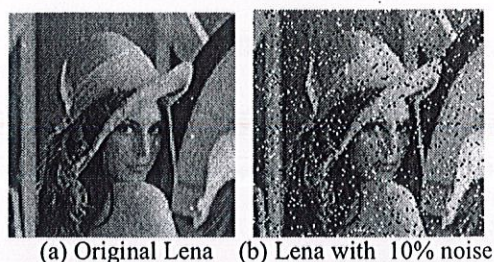


Fig. 14. Comparative reconstructed images using Proposed Neighborhood average filter utilizing DP-PSO and PSO-CNN

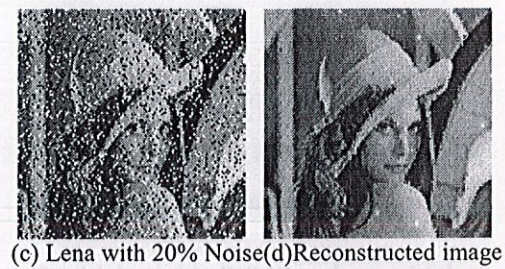
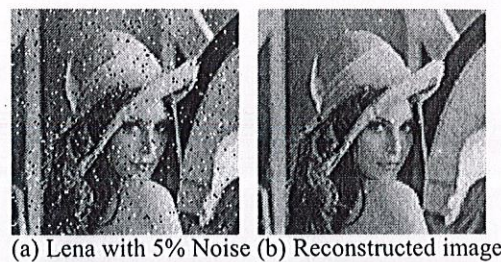


Fig. 15. Noise cancellation results using Proposed Neighborhood average filter utilizing DP-PSO

2. Application 2: Improved Evolutionary Approach to Image Noise Cancellation Utilizing Diminishing Population Particle Swarm Optimization (DP-PSO)

The code of “An Evolutionary Approach to Image Noise Cancellation Utilizing Diminishing Population Particle Swarm Optimization (DP-PSO)” has been optimized by avoiding the unnecessary computations in the cases where the neighboring pixels of the pixel under consideration in a window are all corrupted, which gives us nothing to work upon to remove the noise.

The corruption of images by salt and pepper noise in image acquisition and transmission is unavoidable and has attracted the attention of many researchers over the past decade. This type of noise is introduced by analog-to-digital converter errors, dead pixels or bit errors in transmission. Hence noise removal becomes an important task to perform before doing any subsequent image processing. Previous median-based filters were typically implemented invariantly across an image. Hence they exhibited the tendency to modify both the noisy pixels as well as the undisturbed good pixels.

A. Improved DP-PSO Algorithm

This algorithm has been developed for removing noise from images corrupted with homogeneous salt and pepper noise. Salt and pepper noise corrupts a pixel by modifying its pixel value with either a very high pixel value or with a very low pixel value. In the case of salt and pepper noise, either a value of 0 (low pixel value) is introduced in the pixels or a value of 255 (high pixel value) is introduced in the pixels. The 255 and 0 pixel values correspond to salt noise and pepper noise respectively. The neighborhood average filter finds the average pixel value of all the non impulse pixels i.e. the noise free pixels, in a neighborhood window of $m \times m$ (where m is an integer greater than 2).

For generating the results shown we have set m to 3 which yields PSNR values which are better than the PSNR values generated by setting m to any other higher value. This value of the neighborhood window size has been found by running a number of simulations of our algorithm on different test images. Also setting m to 3 implies that we are always working with a small number of particles at a time. This gives us a small neighborhood in which we can find the optimum solution. In a digital image the data is smooth varying, meaning that the pixel values of nearby pixels will not change suddenly and abruptly. Hence by finding the average value of the noise free pixels in a neighborhood using neighborhood average filter we get a pixel value which is to be used as a target by the particles of the swarm.

Each noisy pixel of the neighborhood is considered as a particle constituting the swarm population and is given an initial velocity and position. These velocity and position attributes are then updated by the DP-PSO algorithm and hence each noisy pixel accelerates and converges towards the average pixel value of the neighborhood in each iteration.

TABLE VII
THE INITIALIZED COEFFICIENTS OF THE DP-PSO

| Coefficients | Value |
|----------------------|-------|
| w | 0.4 |
| Number of iterations | 5 |
| c1 | 1 |
| c2 | 1 |

The distance measure from the neighborhood average value serves as the objective function. When the pixel particle attains the *lBest* value of the neighborhood it is subsequently removed from the swarm. The neighborhood average filter has been used iteratively through the implementation of DP-PSO for removing the noise from the corrupted image. Hence the neighborhood average filter has been optimized by DP-PSO. The steps of the proposed algorithm are given in the pseudocode below.

We first initialize the coefficients of the DP-PSO approach. Table I gives the initialized coefficients of DP-PSO.

B. Pseudocode for each 3*3 neighbourhood

```

psoiterations = 5;           % No of PSO iterations
x = y = 3                   %Dimensions of the Neighborhood
w = 0.4

For r_index = 1 to x
  For c_index = 1 to y
    Calculate average pixel value of the
    neighborhood disregarding noisy pixels
  End
End

% PSO Initialization of Position and Velocity

```


% Particles of the swarm are identified as the pixels which are containing noise

For r_index = 1 to x

For c_index = 1 to y

If pixel is a noisy pixel

flag (r_index , c_index) = 1 %Set flag to identify particles in swarm

v (r_index , c_index) = 0 %For each particle

in the swarm set initial velocity

p (r_index , c_index) = current pixel value % Set initial position

End

End

End

% Initialization of pBest and lBest values

For r_index = 1 to x

For c_index = 1 to y

If (flag (r_index , c_index) = 1)

pBest (r_index , c_index) =current pixel value %Set

pBest value for each particle

End

End

End

lBest = average

% PSO Iterations

For loop = 1 to psoiterations

For r_index = 1 to x

For c_index = 1 to y

If (flag(r_index , c_index) = 1)

*v(r_index , c_index) = w*v (r_index , c_index) +*

*c1*r1*(pBest(r_index , c_index) - p(r_index , c_index)) +*

*c2*r2*(lBest-p(r_index , c_index)*

p(r_index , c_index) = p(r_index , c_index) +

v(r_index , c_index)

If (p(r_index , c_index) - lBest<10) &&

(p(r_index , c_index)-lBest>-10)

% If the pixel has been corrected force convergence

Set flag(r_index , c_index) = 0

% Remove this particle from the swarm as its pixel

value has been corrected and it has converged

pBest (r_index , c_index) = p(r_index , c_index)

End

End

End

End

End

C. Simulation And Results

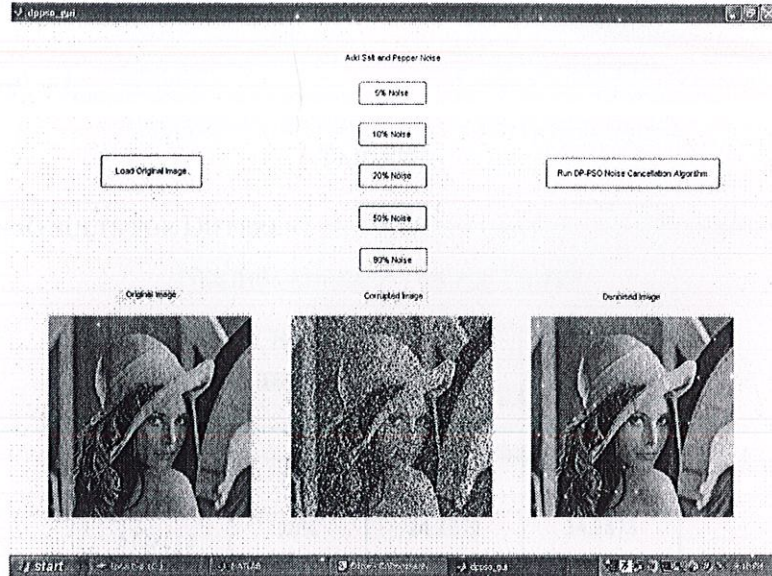


Fig. 16. DP-PSO Algorithm Simulator

In the simulation of the DP-PSO algorithm for noise removal we have encoded the current pixel grayscale values as the positions of the particles of the swarm. These positions are corrected by the DP-PSO algorithm. The filtered value of the neighborhood determined by the neighborhood average filter is the lBest value of the neighborhood. The position and velocity equations of the DP-PSO algorithm are then used in the iterations of the algorithm to update the positions of the particles of the swarm. In each iteration of the DP-PSO algorithm the particles try to improve their fitness values. Hence in this way the algorithm has been and different simulations have been run on different test images of different resolutions and having different levels of noise.

As the salt and pepper noise is homogeneous we have an equal number of pixels corrupted with salt noise and pepper noise. If the level of noise is 30% then 15% of particles are corrupted with salt noise and the other 15% are corrupted with pepper noise.

The objective value calculated by the objective function is given as obj_i . naf_n is the filtered value determined by the neighborhood average filter. The pixel value of pixel

under consideration is given by $i_{x,y}$. The objective function used in the algorithm is given as under

$$obj_i = naf_n - i_{x,y}. \quad (39)$$

TABLE VIII
SIMULATION RESULTS FOR DIFFERENT NOISE DENSITIES FOR OUR PROPOSED OPTIMIZED
NEIGHBORHOOD AVERAGE FILTER

| Test Image | Noise Density | MSE | PSNR (in dB) |
|-----------------|---------------|----------|--------------|
| Lena 512x512 | 5% | 5.9862 | 40.3593 |
| | 10% | 11.5738 | 37.4961 |
| | 20% | 24.2272 | 34.2878 |
| | 50% | 65.5361 | 29.9660 |
| | 80% | 204.8423 | 25.0166 |



(a) Original Lena - 128x128



(b) Lena with 10% noise



(c) Proposed algorithm



(d) PSO-CNN algorithm

Fig. 17. Comparative reconstructed images using Proposed Neighborhood average filter utilizing DP-PSO and PSO-CNN



(a) Lena with 5% Noise



(b) Reconstructed image



(c) Lena with 20% Noise



(d) Reconstructed image

Fig. 18. Noise cancellation results on Lena 128*128 image using Proposed Neighborhood average filter utilizing DP-PSO



(a) Lena with 20% Noise



(b) Reconstructed image



(c) Lena with 50% Noise



(d) Reconstructed image

Fig. 19. Noise cancellation results on Lena 512*512 image using Proposed Neighborhood average filter utilizing DP-PSO

CHAPTER 4

CELLULAR NEURAL NETWORKS

4.1 Introduction

“A neural network is an interconnected assembly of simple processing elements, *units* or *nodes*, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the inter-unit connection strengths, or *weights*, obtained by a process of adaptation to, or *learning* from, a set of training patterns.”

Leon O. Chua AND Yang first introduced the concept of the Cellular Neural Network in 1988 at IEEE transactions on Circuits and Systems.

A **Cellular Neural Network (CNN)**, also known as **Cellular Nonlinear Network**, is an array of dynamical systems (cells) with local interconnections only. The structure of cellular neural networks is similar to that found in cellular automata; i.e., any cell in a cellular neural network is connected only to its neighbor cells. Adjacent cells can interact *directly* with each other. Cells not directly connected together may affect each other indirectly because of the propagation effects of the continuous-time dynamics of the network. Several configurations for cell arrangement exist; however, the most popular is the two-dimensional CNNs organized in an eight-neighbor rectangular grid. Each cell has an input, a state, and an output, and it interacts directly only with the cells within its *radius of neighborhood* r : when $r = 1$, which is a common assumption, the neighborhood includes the cell itself and its eight nearest cells. By varying the values of the connections among cells (i.e., its interaction weights), a CNN can present a large number of dynamics, as proven by Gilli et al. (2002).

Image Processing is the most widespread among the numerous applications of CNNs, (for instance vision systems based on CNN computers are commercially available.

CNNs can be defined as “2D or 3D arrays of mainly locally connected nonlinear dynamical systems called cells, whose dynamics are functionally determined by a small set of parameters which control the cell interconnection strength” (Chua). These

parameters determine the connection pattern, and are collected into the so-called *cloning templates*, which, once determined, define the processing of the whole structure.

Basic Characteristics of the CNN

- The CNN can be defined as an $M \times N$ type array of identical cells arranged in a rectangular grid. Each cell is locally connected to its 8 nearest surrounding neighbors.

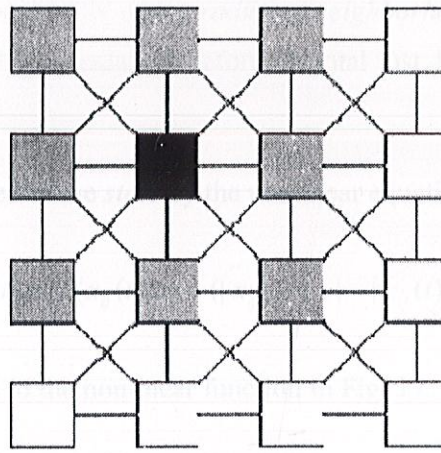


Fig. 20. Two-dimensional CNNs organized in an eight-neighbor rectangular grid.

- Each cell $C(i, j)$ is characterized by u_{ij} , y_{ij} and x_{ij} being the *input*, the *output* and the *state* variable of the cell respectively.
- The CNN dynamics is described by a system of nonlinear differential equations. Using the simplest first-order cell dynamics and linear interactions, the state equation of a cell in position (i, j) is as follows:

$$\frac{dx_{i,j}(t)}{dt} = -x_{ij}(t) + \sum_{(k,l) \in N(i,j)} A(i,j;k,l) \cdot y_{kl}(t) + \sum_{(k,l) \in N(i,j)} B(i,j;k,l) \cdot u_{kl} + z(i,j;k,l) \quad (40)$$

where the indices k and l denote a generic cell belonging to the neighborhood $N(i,j)$ of the cell in position (i,j) . All variables are continuous and z is the *threshold* (bias) of the cell $C(i,j)$. In general, the

state of each cell, and hence its output, depends only on the input and the output of its neighbor cells, and the initial state of the network.

- The set of matrices and the threshold $\{A, B, z\}$, which contain the weights of the neural/nonlinear network, is called *cloning template* and it defines the operation performed by the network. When the values of the *cloning template* do not depend on the position of the cell, the CNN is called *space-invariant*. In this case, the dynamic behavior of the network depends only on a few parameters; for instance, for a two-dimensional CNN with a *radius of neighborhood* $r=1$, A and B are 3×3 matrices, while 'z' is a scalar; therefore, in total just 19 numbers determines the CNN dynamics.
- The *output* is related to the *state* by the nonlinear equation:

$$y_{ij}(t) = f(x_{ij}(t)) = \frac{1}{2}(|x_{ij}(t) + 1| - |x_{ij}(t) - 1|) \quad (41)$$

which corresponds to the nonlinear function in Fig. 17.

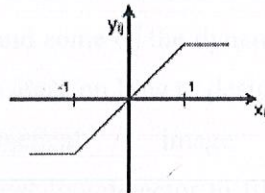


Fig. 21. Nonlinear Function.

The functional model of the CNN architecture

- The output of a CNN model simulation is the final state reached by the network after evolving from an initial state under the influence of a specific input and boundary conditions. The block diagram in Fig 3 shows the state-transition and output of a single cell.
- In the most general case, the final state of one cell can be described by the following equation:
$$x(t) = x(t_0) + \int_{t_0}^t \mathcal{L}(\tau) d\tau = x(t_0) + \int_{t_0}^t f(x(\tau)) d\tau \quad (42)$$

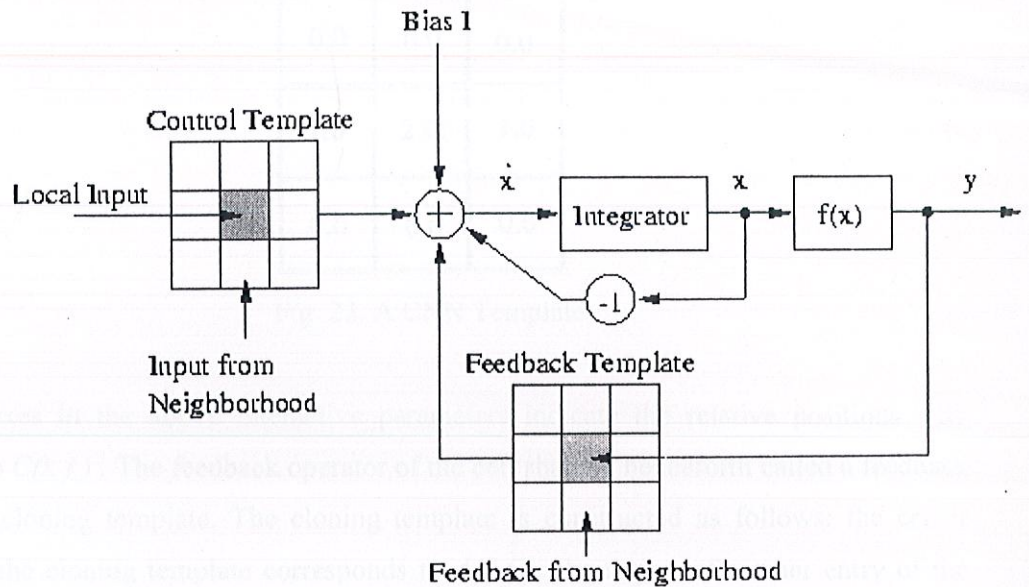


Fig. 22. Functional Model of CNN Architecture

4.2 A Simple Example

This example will help us to understand some of the dynamic behavior of cellular neural networks and to derive some intuitive ideas on how to design cellular neural networks for solving a specific practical image processing problem. Suppose we wish to design a horizontal line detector to filter out the horizontal lines in the input image by using a cellular neural network. In order to simplify our analysis, we have chosen a very simple dynamic rule for this "horizontal line detector" circuit. The circuit element parameters of the cell $C(i, j)$ are chosen as follows for a 3×3 neighborhood system:

$$(\text{Bias}) I = 0$$

We can code the feedback operators $A(i, j; k, l)$ as shown

$$A(-1, -1) = A(-1, 0) = A(-1, 1) = 0$$

$$A(0, 0) = 2$$

$$A(0, -1) = A(0, 1) = 1$$

$$A(1, -1) = A(1,0) = A(1,1) = 0$$

| | | |
|-----|-----|-----|
| 0.0 | 0.0 | 0.0 |
| 1.0 | 2.0 | 1.0 |
| 0.0 | 0.0 | 0.0 |

Fig. 23. A CNN Template

The indexes in the above interactive parameters indicate the relative positions with respect to $C(i, j)$. The feedback operator of the cell shall be henceforth called a feedback operator cloning template. The cloning template is constructed as follows: the center entry of the cloning template corresponds to $A(0,0)$; the upper left corner entry of the cloning template corresponds to $A(-1, -1)$; the lower right-hand corner entry of the cloning template corresponds to $A(1,1)$; and so forth. We can observe that it is extremely convenient and clear to characterize the interactions of a cell with its neighbors by means of a cloning template. Note that we have also chosen the control operator $B(i, j; k, l) = 0$ for all i, j, k , and l in this circuit. Hence from the corresponding state equation that shall be generated we can observe that the derivative of the pixel values depends on their left and right neighbors and themselves, but not on the upper and lower neighbors. This particular dynamic rule will therefore enhance the detection of horizontal lines in the original image.

CHAPTER 5

PROJECT SIMULATOR

5.1 Building Graphical User Interfaces using GUIDE in Matlab

Introduction

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance with the help of inbuilt controls such as pushbuttons, list boxes, sliders, menus, and so forth. It can also make it easier to adjust parameters and to visualize the program. Additionally, a graphical user interface provides the user with a familiar environment in which to work so that he or she can concentrate on using the application rather than on the mechanics involved in doing things. GUI-based programs respond to input events and are hence said to be *event driven*.

The three principal elements required to create a MATLAB Graphical User Interface are:

- 1. Components.**

Each item on a MATLAB GUI (pushbuttons, labels, edit boxes, etc.) is a graphical component. The types of components include graphical controls (pushbuttons, edit boxes, lists, sliders, etc.), static elements (frames and text strings), menus, and axes. Graphical controls and static elements are created by the function `uicontrol`, and menus are created by the functions `uimenu` and `uicontextmenu`. Axes, which are used to display graphical data, are created by the function `axes`.

- 2. Figures.**

The components of a GUI are arranged within a figure, which is a window on the computer screen.

- 3. Callbacks.**

A mouse click or a key press is an event, and the MATLAB program must respond to each event if the program is to perform its function. For example, if a user clicks on a button, that event must cause the MATLAB code that implements the function of the button to be executed. The code executed in response to an

event is known as a call back. There must be a callback to implement the function of each graphical component on the GUI. The basic GUI elements are summarized in Table 1.

GUIDE is Matlab's Graphics User Interface (GUI) Design Environment

- GUIDE stores GUIs in two files, which are generated the first time the GUI is saved or executed:
 - ❖ .fig file - contains a complete description of the GUI figure layout and the components of the GUI. Changes to this file are made in the Layout Editor
 - ❖ .m file - contains the code that controls the GUI. Here the callbacks can be programmed using the M-file Editor.

Typical stages of creating a GUI are:

1. Designing the GUI
2. Laying out the GUI – Using the Layout Editor
3. Programming the GUI – Writing callbacks in the M-file Editor
4. Saving and Running the GUI

Basic Controls

- ❖ Axes: Creates axes graphics object.
- ❖ Static Text: text that is stuck on the screen and which cannot be edited.
- ❖ Edit Box: a white box for typing information into.
- ❖ Button: performs an action when user clicks on it.

The basic steps required to create a MATLAB GUI are:

1. Decide what elements are required for the GUI and the corresponding functionality associated with each element. Make a rough layout of the components by hand on a piece of paper.
2. Use Matlab's guide (GUI Development Environment) to layout the Components on a figure. Adjust size of the figure and the alignment and spacing of components using the tools built in the guide.
3. Use a MATLAB tool called the Property Inspector (built in the guide) to give each component a name (a "tag") and to set the characteristics of each component, such as its color, the text it displays, and so on.

4. Save the figure to a file.
5. Write code to implement the behavior associated with each callback function

5.2 The Simulator

Salient Features

- One main interface
- Main interface provides access to the simulators of all the developed algorithms
- MSE and PSNR calculator
- Easy GUI to use
- Has File Options in the File menu
 - Open
 - Print
 - Close
- Can run multiple GUI's simultaneously from the main interface
- Helps in running an algorithm and simultaneously evaluating it using the MSE and PSNR Calculator
- Can run different algorithms simultaneously and compare them

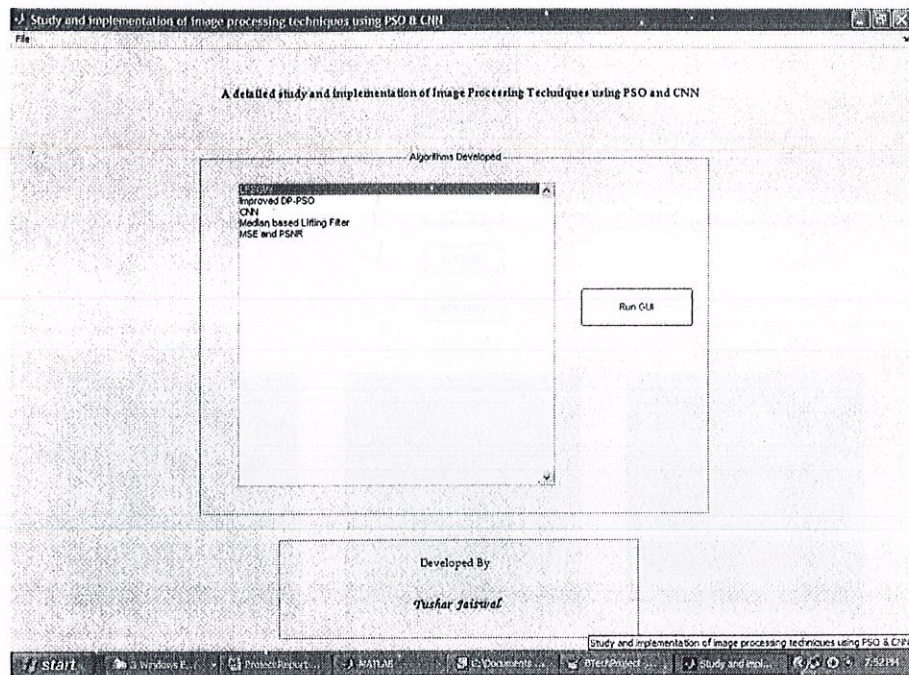
Salient Features of MSE and PSNR Calculator

- Uses standard formulae to calculate MSE and PSNR

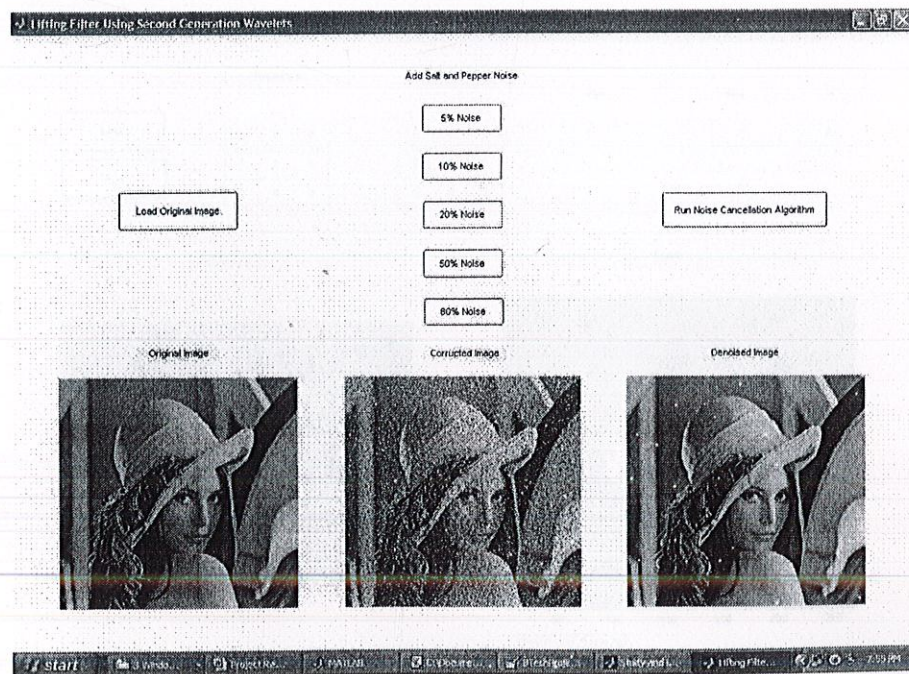
$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \|I(i, j) - R(i, j)\|^2 \quad (43)$$

$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \quad (44)$$

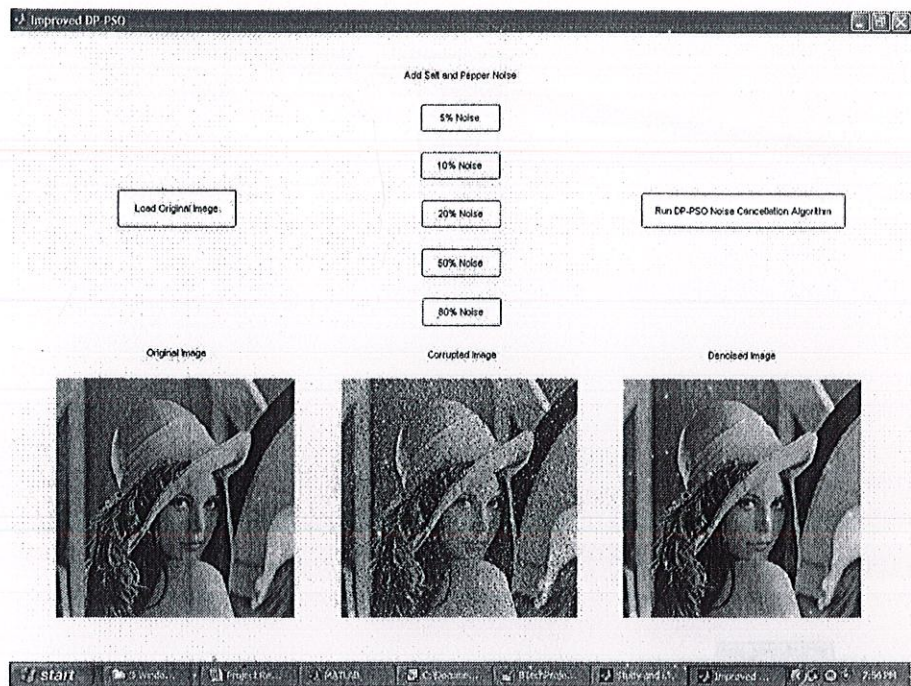
- Even works for colour images
- We just need an original and a corrupted image to compare
- Has a simple and intuitive interface to use



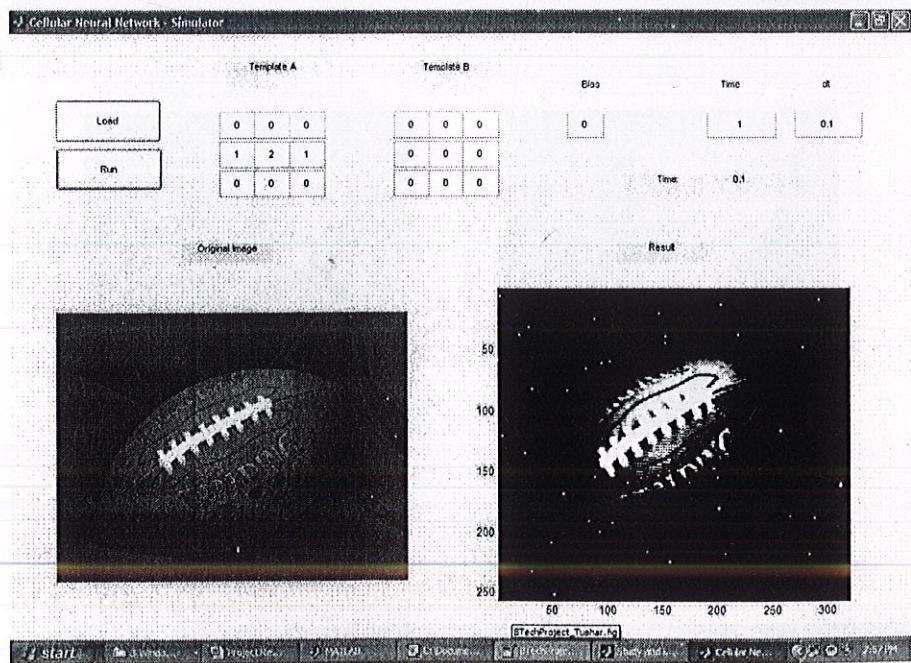
Main Interface



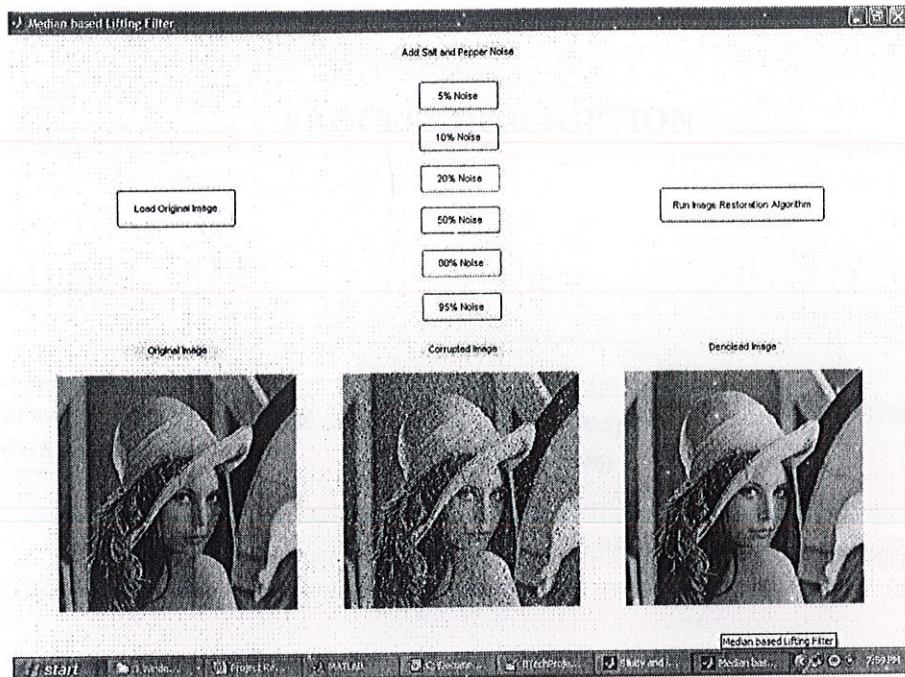
LFSGW Simulator



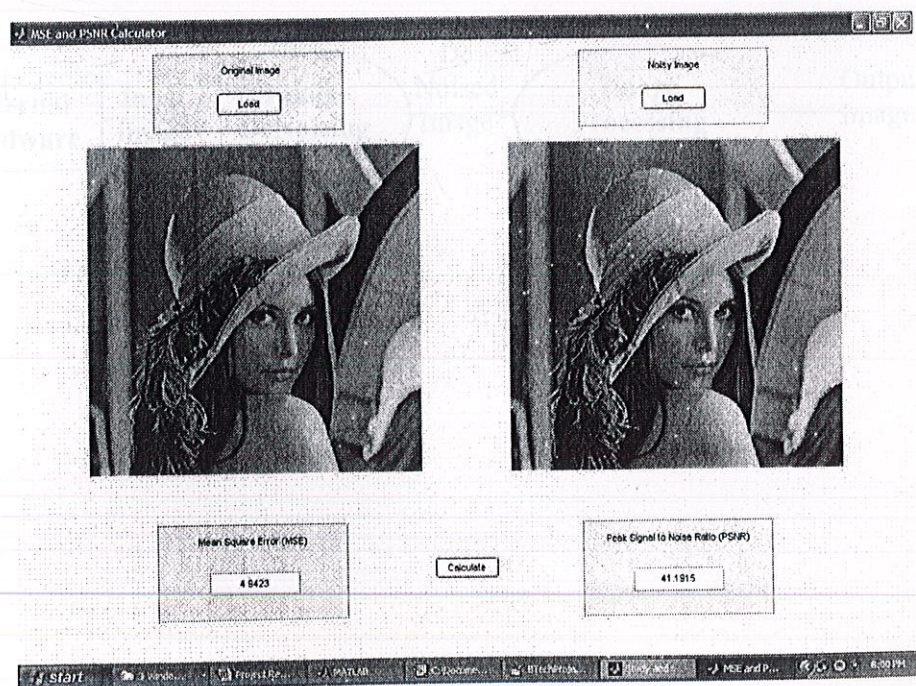
Improved DP-PSO Simulator



CNN Line Detector Simulator



Median based Lifting Filter Simulator

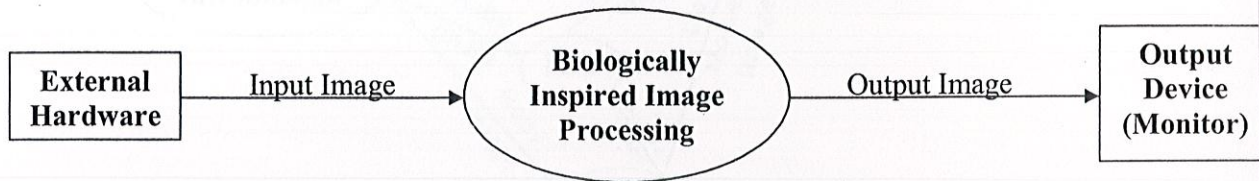


MSE and PSNR Calculator

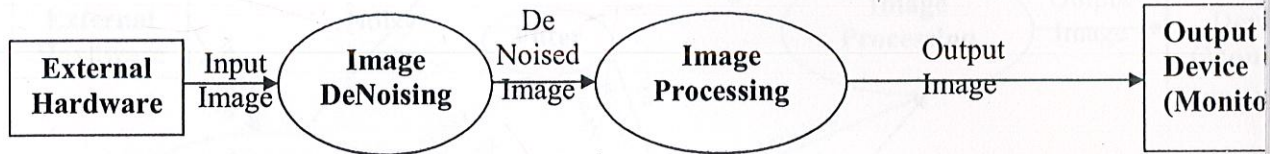
CHAPTER 6

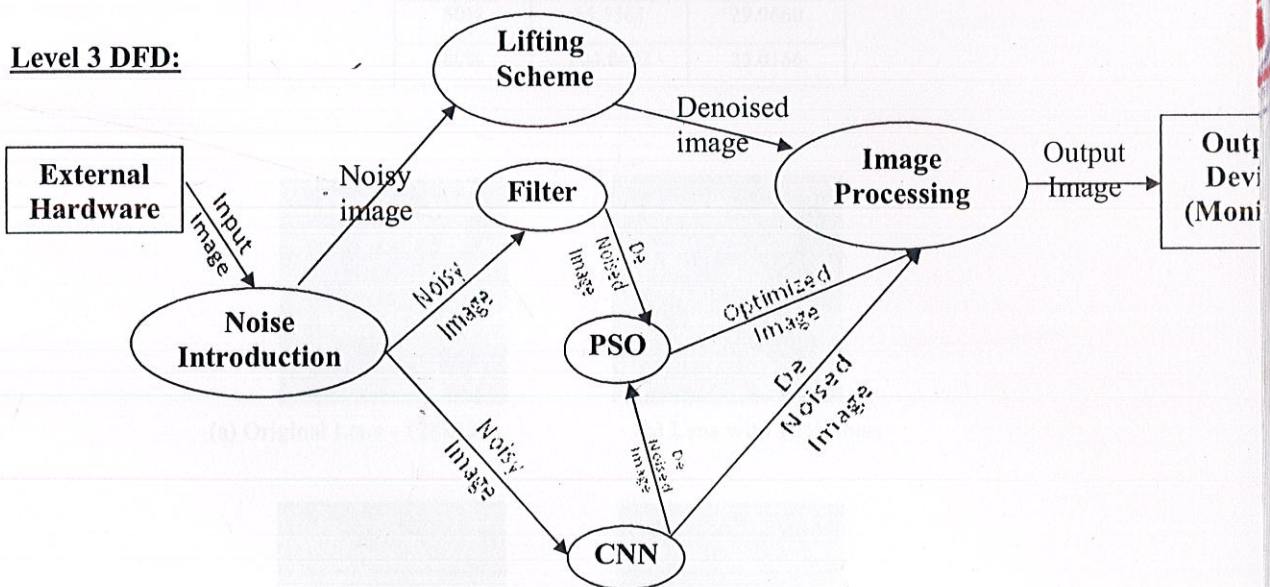
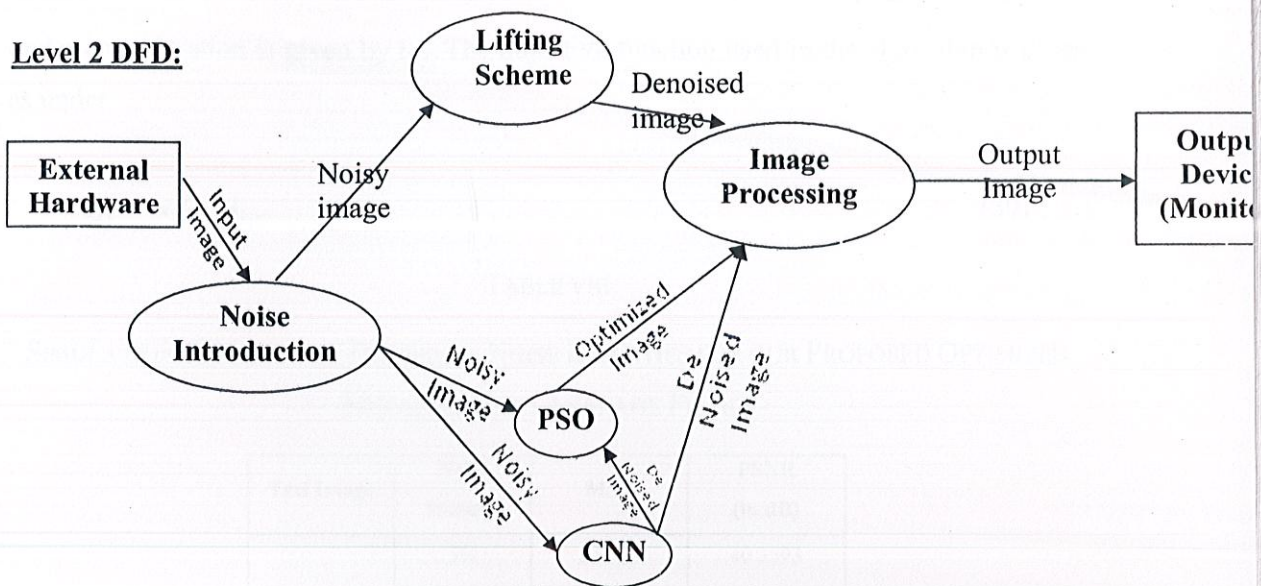
PROCESS DESCRIPTION

Level 0 DFD:



Level 1 DFD:





under consideration is given by $i_{x,y}$. The objective function used in the algorithm is given as under

$$obj_i = naf_n - i_{x,y}. \quad (39)$$

TABLE VIII
SIMULATION RESULTS FOR DIFFERENT NOISE DENSITIES FOR OUR PROPOSED OPTIMIZED
NEIGHBORHOOD AVERAGE FILTER

| Test Image | Noise Density | MSE | PSNR (in dB) |
|-----------------|---------------|----------|--------------|
| Lena 512x512 | 5% | 5.9862 | 40.3593 |
| | 10% | 11.5738 | 37.4961 |
| | 20% | 24.2272 | 34.2878 |
| | 50% | 65.5361 | 29.9660 |
| | 80% | 204.8423 | 25.0166 |



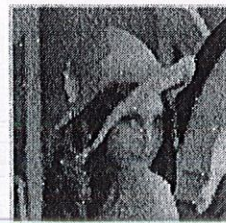
(a) Original Lena - 128x128



(b) Lena with 10% noise



(c) Proposed algorithm



(d) PSO-CNN algorithm

Fig. 17. Comparative reconstructed images using Proposed Neighborhood average filter utilizing DP-PSO and PSO-CNN



(a) Lena with 5% Noise



(b) Reconstructed image



(c) Lena with 20% Noise

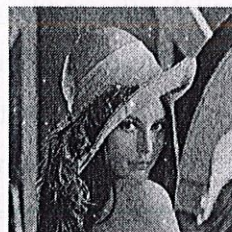


(d) Reconstructed image

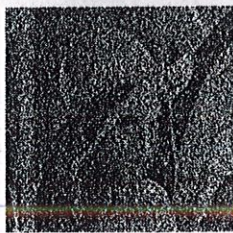
Fig. 18. Noise cancellation results on Lena 128*128 image using Proposed Neighborhood average filter utilizing DP-PSO



(a) Lena with 20% Noise



(b) Reconstructed image



(c) Lena with 50% Noise



(d) Reconstructed image

Fig. 19. Noise cancellation results on Lena 512*512 image using Proposed Neighborhood average filter utilizing DP-PSO

CHAPTER 4

CELLULAR NEURAL NETWORKS

4.1 Introduction

“A neural network is an interconnected assembly of simple processing elements, *units* or *nodes*, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the inter-unit connection strengths, or *weights*, obtained by a process of adaptation to, or *learning* from, a set of training patterns.”

Leon O. Chua AND Yang first introduced the concept of the Cellular Neural Network in 1988 at IEEE transactions on Circuits and Systems.

A Cellular Neural Network (CNN), also known as Cellular Nonlinear Network, is an array of dynamical systems (cells) with local interconnections only. The structure of cellular neural networks is similar to that found in cellular automata; i.e., any cell in a cellular neural network is connected only to its neighbor cells. Adjacent cells can interact *directly* with each other. Cells not directly connected together may affect each other indirectly because of the propagation effects of the continuous-time dynamics of the network. Several configurations for cell arrangement exist; however, the most popular is the two-dimensional CNNs organized in an eight-neighbor rectangular grid. Each cell has an input, a state, and an output, and it interacts directly only with the cells within its *radius of neighborhood r* : when $r = 1$, which is a common assumption, the neighborhood includes the cell itself and its eight nearest cells. By varying the values of the connections among cells (i.e., its interaction weights), a CNN can present a large number of dynamics, as proven by Gilli et al. (2002).

Image Processing is the most widespread among the numerous applications of CNNs, (for instance vision systems based on CNN computers are commercially available.

CNNs can be defined as “2D or 3D arrays of mainly locally connected nonlinear dynamical systems called cells, whose dynamics are functionally determined by a small set of parameters which control the cell interconnection strength” (Chua). These

parameters determine the connection pattern, and are collected into the so-called *cloning templates*, which, once determined, define the processing of the whole structure.

Basic Characteristics of the CNN

- The CNN can be defined as an $M \times N$ type array of identical cells arranged in a rectangular grid. Each cell is locally connected to its 8 nearest surrounding neighbors.

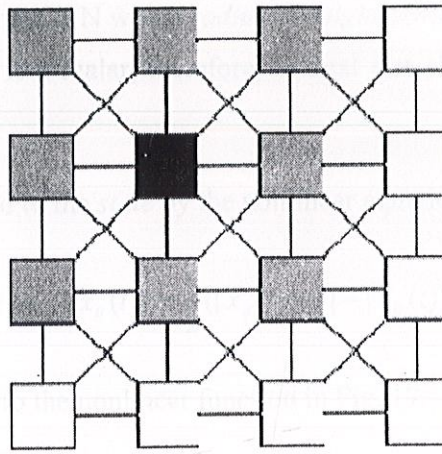


Fig. 20. Two-dimensional CNNs organized in an eight-neighbor rectangular grid.

- Each cell $C(i, j)$ is characterized by u_{ij} , y_{ij} and x_{ij} being the *input*, the *output* and the *state* variable of the cell respectively.
- The CNN dynamics is described by a system of nonlinear differential equations. Using the simplest first-order cell dynamics and linear interactions, the state equation of a cell in position (i, j) is as follows:

$$\frac{dx_{i,j}(t)}{dt} = -x_{ij}(t) + \sum_{(k,l) \in N(i,j)} A(i,j;k,l) \cdot y_{kl}(t) + \sum_{(k,l) \in N(i,j)} B(i,j;k,l) \cdot u_{kl} + z(i,j;k,l) \quad (40)$$

where the indices k and l denote a generic cell belonging to the neighborhood $\mathcal{N}(i,j)$ of the cell in position (i,j) . All variables are continuous and z is the *threshold* (bias) of the cell $C(i,j)$. In general, the

state of each cell, and hence its output, depends only on the input and the output of its neighbor cells, and the initial state of the network.

- The set of matrices and the threshold $\{A, B, z\}$, which contain the weights of the neural/nonlinear network, is called *cloning template* and it defines the operation performed by the network. When the values of the *cloning template* do not depend on the position of the cell, the CNN is called *space-invariant*. In this case, the dynamic behavior of the network depends only on a few parameters; for instance, for a two-dimensional CNN with a *radius of neighborhood* $r=1$, A and B are 3×3 matrices, while 'z' is a scalar; therefore, in total just 19 numbers determines the CNN dynamics.
- The *output* is related to the *state* by the nonlinear equation:

$$y_{ij}(t) = f(x_{ij}(t)) = \frac{1}{2}(|x_{ij}(t) + 1| - |x_{ij}(t) - 1|) \quad (41)$$

which corresponds to the nonlinear function in Fig. 17.

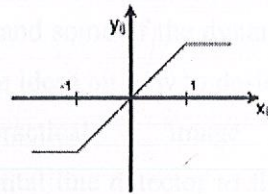


Fig. 21. Nonlinear Function.

The functional model of the CNN architecture

- The output of a CNN model simulation is the final state reached by the network after evolving from an initial state under the influence of a specific input and boundary conditions. The block diagram in Fig 3 shows the state-transition and output of a single cell.
- In the most general case, the final state of one cell, can be described by the following equation:
$$x(t) = x(t_0) + \int_{t_0}^t \dot{x}(\tau) d\tau = x(t_0) + \int_{t_0}^t f(x(\tau)) d\tau \quad (42)$$

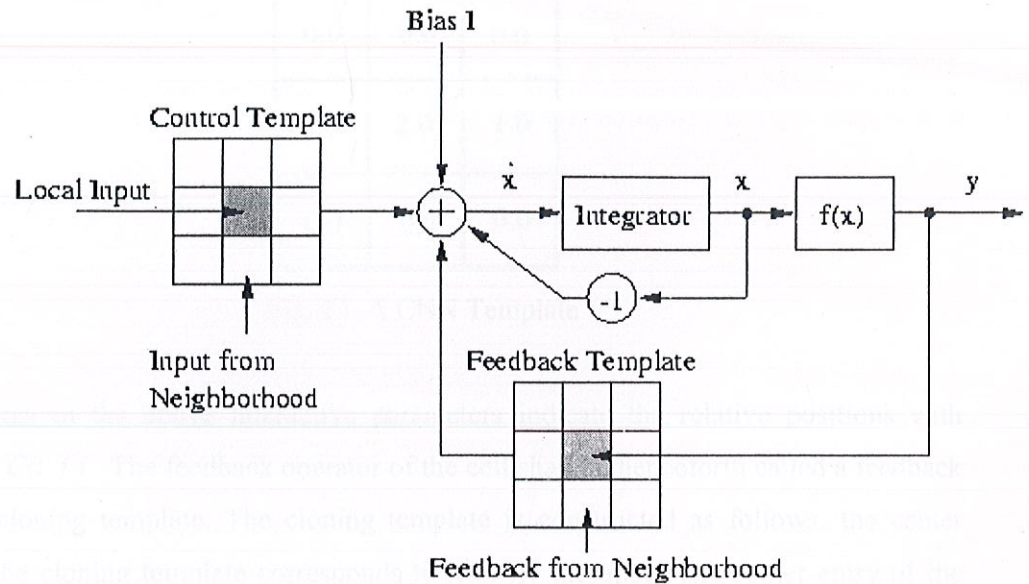


Fig. 22. Functional Model of CNN Architecture

4.2 A Simple Example

This example will help us to understand some of the dynamic behavior of cellular neural networks and to derive some intuitive ideas on how to design cellular neural networks for solving a specific practical image processing problem. Suppose we wish to design a horizontal line detector to filter out the horizontal lines in the input image by using a cellular neural network. In order to simplify our analysis, we have chosen a very simple dynamic rule for this "horizontal line detector" circuit. The circuit element parameters of the cell $C(i, j)$ are chosen as follows for a 3 x 3 neighborhood system:

$$(Bias) I = 0$$

We can code the feedback operators $A(i, j; k, l)$ as shown

$$A(-1, -1) = A(-1, 0) = A(-1, 1) = 0$$

$$A(0, 0) = 2$$

$$A(0, -1) = A(0, 1) = 1$$

$$A(1, -1) = A(1,0) = A(1,1) = 0$$

| | | |
|-----|-----|-----|
| 0.0 | 0.0 | 0.0 |
| 1.0 | 2.0 | 1.0 |
| 0.0 | 0.0 | 0.0 |

Fig. 23. A CNN Template

The indexes in the above interactive parameters indicate the relative positions with respect to $C(i, j)$. The feedback operator of the cell shall be henceforth called a feedback operator cloning template. The cloning template is constructed as follows: the center entry of the cloning template corresponds to $A(0,0)$; the upper left corner entry of the cloning template corresponds to $A(-1, -1)$; the lower right-hand corner entry of the cloning template corresponds to $A(1,1)$; and so forth. We can observe that it is extremely convenient and clear to characterize the interactions of a cell with its neighbors by means of a cloning template. Note that we have also chosen the control operator $B(i, j, k, l) = 0$ for all i, j, k , and l in this circuit. Hence from the corresponding state equation that shall be generated we can observe that the derivative of the pixel values depends on their left and right neighbors and themselves, but not on the upper and lower neighbors. This particular dynamic rule will therefore enhance the detection of horizontal lines in the original image.

CHAPTER 5

PROJECT SIMULATOR

5.1 Building Graphical User Interfaces using GUIDE in Matlab

Introduction

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance with the help of inbuilt controls such as pushbuttons, list boxes, sliders, menus, and so forth. It can also make it easier to adjust parameters and to visualize the program. Additionally, a graphical user interface provides the user with a familiar environment in which to work so that he or she can concentrate on using the application rather than on the mechanics involved in doing things. GUI-based program respond to input events and are hence said to be *event driven*.

The three principal elements required to create a MATLAB Graphical User Interface are:

- 1. Components.**

Each item on a MATLAB GUI (pushbuttons, labels, edit boxes, etc.) is a graphical component. The types of components include graphical controls (pushbuttons, edit boxes, lists, sliders, etc.), static elements (frames and text strings), menus, and axes. Graphical controls and static elements are created by the function `uicontrol`, and menus are created by the functions `uimenu` and `uicontextmenu`. Axes, which are used to display graphical data, are created by the function `axes`.

- 2. Figures.**

The components of a GUI are arranged within a figure, which is a window on the computer screen.

- 3. Callbacks.**

A mouse click or a key press is an event, and the MATLAB program must respond to each event if the program is to perform its function. For example, if a user clicks on a button, that event must cause the MATLAB code that implements the function of the button to be executed. The code executed in response to an

event is known as a call back. There must be a callback to implement the function of each graphical component on the GUI. The basic GUI elements are summarized in Table 1.

GUIDE is Matlab's Graphics User Interface (GUI) Design Environment

- GUIDE stores GUIs in two files, which are generated the first time the GUI is saved or executed:
 - ❖ .fig file - contains a complete description of the GUI figure layout and the components of the GUI. Changes to this file are made in the Layout Editor
 - ❖ .m file - contains the code that controls the GUI. Here the callbacks can be programmed using the M-file Editor.

Typical stages of creating a GUI are:

1. Designing the GUI
2. Laying out the GUI – Using the Layout Editor
3. Programming the GUI – Writing callbacks in the M-file Editor
4. Saving and Running the GUI

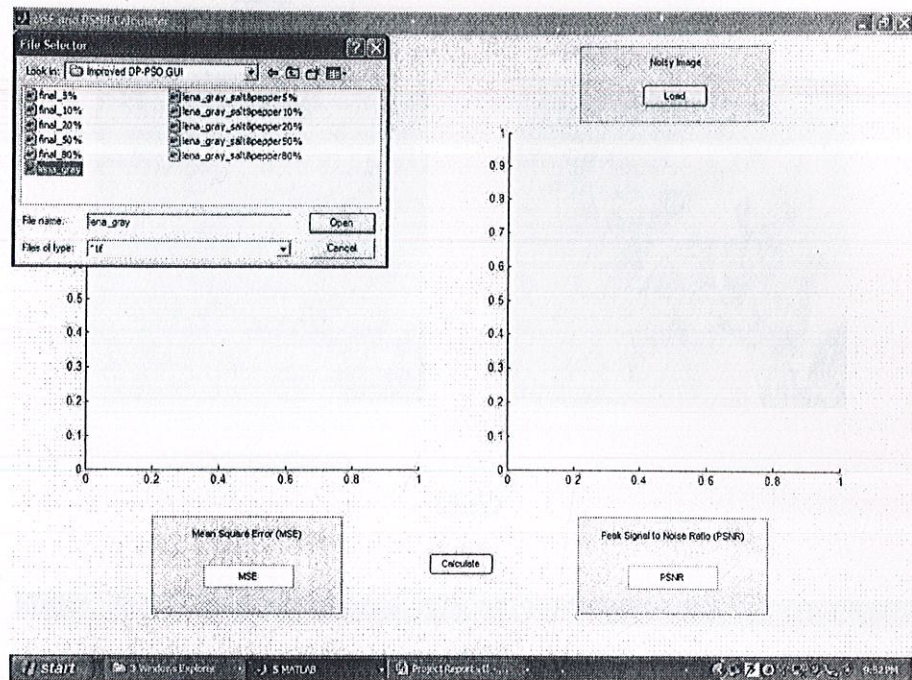
Basic Controls

- ❖ Axes: Creates axes graphics object.
- ❖ Static Text: text that is stuck on the screen and which cannot be edited.
- ❖ Edit Box: a white box for typing information into.
- ❖ Button: performs an action when user clicks on it.

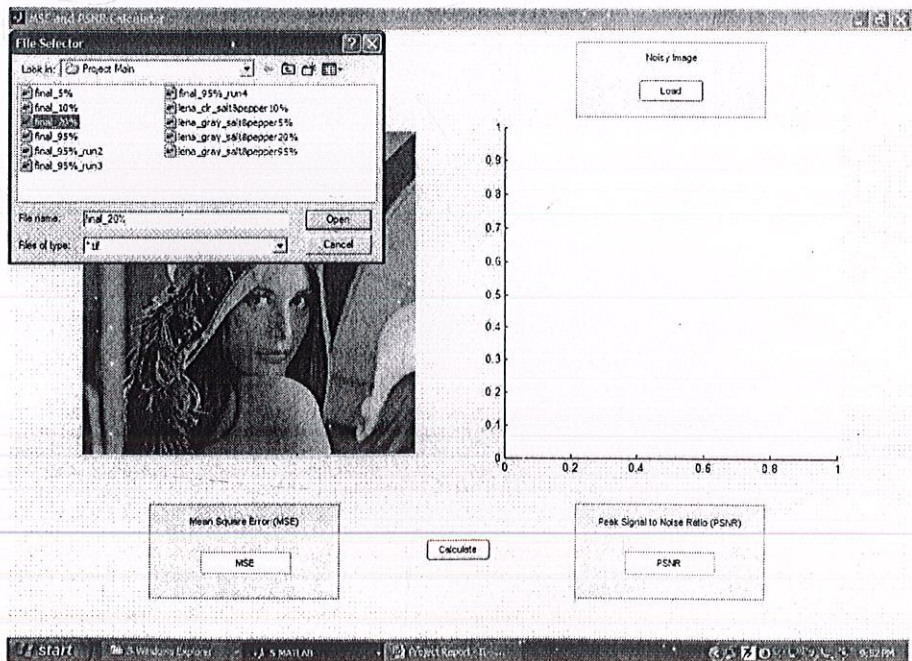
The basic steps required to create a MATLAB GUI are:

1. Decide what elements are required for the GUI and the corresponding functionality associated with each element. Make a rough layout of the components by hand on a piece of paper.
2. Use Matlab's guide (GUI Development Environment) to layout the Components on a figure. Adjust size of the figure and the alignment and spacing of components using the tools built in the guide.
3. Use a MATLAB tool called the Property Inspector (built in the guide) to give each component a name (a "tag") and to set the characteristics of each component, such as its color, the text it displays, and so on.

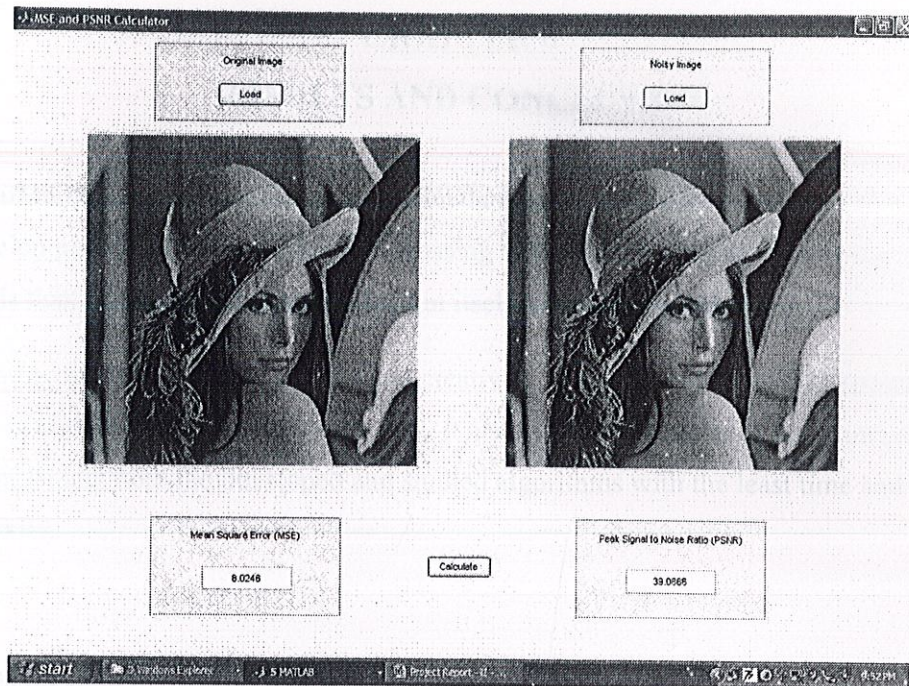
MSE and PSNR Calculator



Loading Original Image



Loading Corrupted Image



Calculating MSE and PSNR

CHAPTER 8

RESULTS AND CONCLUSION

Innovation in the Project: The project stimulates our minds and challenges us by letting us develop novel methods for Image Processing tasks. Also it aims at developing better methods than the existing methods which in itself is a daunting task.

Thus the project demands innovation and creativity. A strong and up to date knowledge base is of utmost importance in the project. It also requires dexterity in programming so as to implement both the developed and studied algorithms with the least time and space complexity.

Main Achievements: As of now three algorithms have been developed for Image Noise Cancellation. The research papers which introduce those algorithms are:

- 1) Image Noise Cancellation by Lifting Filter using Second Generation Wavelets (LFSGW) – Published in IEEE Xplore and IEEE CS Digital Library through the Proceedings of IEEE International Conference on Advances in Recent Technologies in Communication and Computing, ARTCom 2009, October 27th -28th 2009, Kottayam, Kerala, India.
- 2) Fast Image Restoration by Median Based Lifting Filter using Second Generation Wavelets – Accepted to be published in IEEE International Conference on Industrial Technology (ICIT), 14-17th March 2010, Viña del Mar - Valparaiso, Chile.
- 3) An Evolutionary Approach to Image Noise Cancellation Utilizing Diminishing Population Particle Swarm Optimization (DP-PSO)

Apart from this GUI's have been developed for simulating the developed algorithms.

Thus the highlight of the project is:

- 1) It is a research based project with complete implementation
- 2) It has its application in the real world
- 3) The developed algorithms are giving great results, which is also appreciated by the research community as can be seen from the acceptance of our papers in reputed conferences.
- 4) The developed GUI's enables us to better understand the algorithms

Hence it can be said that this project has achieved a lot more than what a normal project seeks to achieve through the effective implementation of novel algorithms developed by us and the algorithms which we have studied.

CONTRIBUTION OF THE PROJECT

The project contributes greatly to the Image Processing research community and the Academia. The project may also be of great use to the Industry employing Image processing.

- 1) **Academia:** People who are studying Image Processing will be greatly benefited through this project. The project provides them with a pathway to understand Image Processing, the various Image Processing tasks and different techniques employed in Image Processing. They will be the ones who will benefit the most from the Graphical User Interfaces (GUI's) developed.
- 2) **Image Processing Research Community:** As the project aims at developing novel algorithms for Image Processing tasks such as Image Noise Cancellation the research community can also benefit from it. The algorithms will provide new and efficient ways which are different from the existing works and will try to open up a new frontier for researchers to work with by providing a different approach to solving problems. This is evident from the development of novel Noise Cancellation methods which employ the Lifting Scheme. This approach for Noise Cancellation had not been adopted before.
- 3) **Image Processing Industry:** The algorithms developed can be directly used in the Image Processing Industry or they can be further improved by adding application specific features and then employed in the Image Processing Industry. The developed algorithms can thus be used for Image Processing tasks in Image Processing softwares. They can be used in digital cameras, scanners and other Image Acquisition devices.

BIBLIOGRAPHY

1. MATLAB 7.0 Documentation
2. Rafael C. Gonzalez, *Digital Image Processing*, Prentice Hall, 2006.
3. Rafael C. Gonzalez, *Digital Image Processing Using Matlab*, Prentice Hall, 2006.
4. Jaiswal, T.; Rajesh, S., "Image Noise Cancellation by Lifting Filter Using Second Generation Wavelets (LFSGW)," *Advances in Recent Technologies in Communication and Computing*, 2009. ARTCom '09. International Conference on , vol., no., pp.667-671, 27-28 Oct. 2009
5. Siddavatam Rajesh, K Sandeep and R K Mittal , "A Fast Progressive Image Sampling Using Lifting Scheme And Non-Uniform B-Splines", *Proceedings of IEEE_ International Symposium on Industrial Electronics ISIE -07*, June 4-7, pp. 1645- 1650, Vigo, Spain, 2007.
6. Chuen-Yau Chen and Chih-Wen Hsia, "Image Noise Cancellation by Adaptive Filter with Weight-Training Mechanism (AFWTM)", *In the Proceedings of Information Decision and Control, IDC 07*, 12-14 Feb, pp. 332-335, Adelaide, 2007.
7. W.Swelden, "The Lifting scheme : A custom design construction of biorthogonal wavelets", *Appl. Comput. Harmon. Anal.*, 3(2), pp. 186-200, 1996.
8. W.Swelden, "The Lifting Scheme: A Construction of second generation wavelets", *Technical Report 1995; 6*, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1995.
9. I. Pitas and A. N. Venetsanopoulos, *Nonlinear Digital Filters: Principles and Applications*. Boston, MA: Kluwer, 1990.
10. D. R. K. Brownrigg, "The weighted median filter," *Commun. Ass. Comput. Mach.*, vol. 27, no. 8, pp. 807-818, Aug. 1984.

11. T. Sun and Y. Neuvo, "Detail-preserving median based filters in image processing," *Pattern Recognit. Lett.*, vol. 15, pp. 341–347, Apr. 1994.
12. D. Zhang and Z. Wang, "Impulse noise detection and removal using fuzzy techniques," *Electron. Lett.*, vol. 33, pp. 378–379, Feb. 1997.
13. E. Abreu, M. Lightstone, S. K. Mitra, and K. Arakawa, "A new efficient approach for the removal of impulse noise from highly corrupted images," *IEEE Trans. Image Processing*, vol. 5, pp. 1012–1025, June 1996.
14. Yue-Cheng Chen, Hsin-Chih Wang and Te-Jen Su, "Particle Swarm Optimization for Image Noise Cancellation," *Innovative Computing, Information and Control*, 2006. ICICIC '06. First International Conference on , vol.1, no., pp.587-590, Aug. 30 2006-Sept. 1 2006.
15. Sheng-Fu Liang, Shih-Mao Lu, Jyh-Yeong Chang and Chin-Teng Lin, "A Novel Two-Stage Impulse Noise Removal Technique Based on Neural Networks and Fuzzy Decision," *Fuzzy Systems, IEEE Transactions on* , vol.16, no.4, pp.863-873, Aug. 2008.
16. Nallaperumal, K., Varghese, J., Saudia, S., Mathew, S.P., Krishnaveni, K., Annam, S., "Adaptive Rank-ordered Switching Median Filter for Salt & Pepper Impulse Noise Reduction," *India Conference, 2006 Annual IEEE*, vol., no., pp.1-6, 15-17 Sept. 2006.
17. Zhou Wang and David Zhang, "Progressive Switching Median Filter for the Removal of Impulse Noise from Highly Corrupted Images", *IEEE Transactions on Circuits And Systems—II: Analog And Digital Signal Processing*, Vol. 46, No. 1, January 1999.
18. Wenbin Luo, Member, IEEE, "An Efficient Detail-Preserving Approach for Removing Impulse Noise in Images", *IEEE Signal Processing Letters*, Vol. 13, No. 7, July 2006.

19. Crnojevic, V.; Senk, V.; Trpovski, Z., "Advanced impulse Detection Based on pixel-wise MAD," *Signal Processing Letters, IEEE* , vol.11,no.7, pp. 589-592, July 2004.
20. Gouchol Pok; Jyh-Charn Liu; Nair, A.S., "Selective removal of impulse noise based on homogeneity level information," *Image Processing, IEEE Transactions on* , vol.12, no.1, pp. 85-92, Jan 2003.
21. Tao Chen; Hong Ren Wu, "Application of partition-based median type filters for suppressing noise in images," *IEEE Transactions on Image Processing* , vol.10, no.6, pp.829-836, Jun 2001.
22. R. H. Chan, C. Hu, and M. Nikolova, "An iterative procedure for removing random-valued impulse noise," *IEEE Signal Process. Lett.*, vol. 11, no. 12, pp. 921--924, Dec. 2004.
23. Te-Jen Su, Tzu-Hsiang Lin, Jia-Wei Liu, "Particle Swarm Optimization for Gray-Scale Image Noise Cancellation," *Intelligent Information Hiding and Multimedia Signal Processing, 2008. IHHMSP '08 International Conference on* , vol., no., pp.1459-1462, 15-17 Aug. 2008.
24. Christian Blum, Daniel Merkle, (Eds.) "Swarm Intelligence Introduction and Applications". Natural Computing Series, Springer Verlag, 2008.
25. M. Fatih Tasgetiren, P. N. Suganthan, Quan-Ke Pan, "A Discrete Particle Swarm Optimization Algorithm for the Generalized Traveling Salesman Problem", *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 158 – 167, (2007).
26. Wachowiak, M.P., Smolikova, R., Yufeng Zheng, Zurada, J.M., Elmaghraby, A.S., "An approach to multimodal biomedical image registration utilizing particle swarm optimization," *Evolutionary Computation, IEEE Transactions on* , vol.8, no.3, pp. 289-301, June 2004.

27. H. Talbil, M. C. Batouche, "Particle swarm optimization for image registration", International Conference on Computer Theory and Applications, IEEE Press, Damascus, Syria, April 2004.
28. Wooi-Haw Tan, Rosli Besar, Gouenou Coatrieux and Basel Solaiman, "PSO based parametric object recognition", IEICE Electron. Express, Vol. 5, No. 24, pp.1080-1087, (2008).
29. Yue-Cheng Chen , Hsin-Chih Wang , Te-Jen Su, Particle Swarm Optimization for Image Noise Cancellation, Proceedings of the First International Conference on Innovative Computing, Information and Control, p.587-590, August 30-September 01, 2006.
30. Soudan, B., Saad, M., "An Evolutionary Dynamic Population Size PSO Implementation," Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on , vol., no., pp.1-5, 7-11 April 2008.
31. J. Kenndy and R. C. Eberhart, "Particle Swarm Optimization," Proceedings of IEEE International Conference on Neural Networks, Pp. 1942-1948.(1995).
32. R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in Proc. 6th Int. Symp. Micromachine Human Sci., Nagoya, Japan, 1995, pp. 39-43.
33. Frans Van Den Bergh, A. P. Engelbrecht, "An analysis of particle swarm optimizers", University of Pretoria, Pretoria, South Africa, 2002.
34. L. Chua and L. Yang, "Cellular Neural Networks: Theory," IEEE Trans. on Circuits and Systems, 35(10):1257-1272, 1988.
35. L. Chua and L. Yang, "Cellular Neural Networks: Applications" IEEE Trans. on Circuits and Systems, 35(10):1273:1290, 1988.
36. PSO tutorial by Xiaohui Hu, Ph.D.
37. The Wavelet tutorial by Robi Polikar.

APPENDIX-I

SAMPLE CODES OF PROJECT

Noise Addition Code:

```
k=imread('lena_std.tif');
i=rgb2gray(k);
imwrite(i,'lena_gray.tif','Compression','none')
sizeA=size(i)
p3=0.1;
b = i;
x = rand(sizeA);
b(x < p3/2) = 0; % Minimum value
b(x >= p3/2 & x < p3) = 255; % Maximum (saturated) value
imshow(b)
imwrite(b,'lena_gray_salt&pepper10%.tif','Compression','none')
```

Image Noise Cancellation by Lifting Filter using Second Generation Wavelets

(LFSGW)

```
i=imread('lena_clr_salt&pepper10%.tif');
threshold=10;%Assuming that image data with a value of 0 or 255 will have neighbours
whose values will not vary more than +/-10 from 0 or 255
sizei=size(i);
j=i;
avg=i;
%j=-1*ones(sizei);%giving all the pixels the value -1 indicating it is image data
%avg=-1*ones(sizei);%giving all the pixels the value -1 indicating it is image data
data=i;
for m = 1:sizei(1,1)
    for n = 1:sizei(1,2)
        for o= 1:3
```



```

        if(data(m,n,o)==0||data(m,n,o)==255)
            j(m,n,o)=data(m,n,o);
        end;
    end;
end;
end;
%j=i(data==0 | data==255);%lifting,storing the value of lifted pixel in j
sig=-1*ones(sizei);%significant pixels(pixel value(z)=0 or 255)-noise
insig=-1*ones(sizei);%insignificant pixels(pixel value(z)=0 or 255)-image data
%Top Left Corner Pixel
for o= 1:3
    avg(1,1,o)=uint8((uint16(j(1,2))+uint16(j(2,1))+uint16(j(2,2)))/3);
    if(((j(1,1,o)-avg(1,1,o))>10 || (avg(1,1,o)-j(1,1,o))>10)&&(j(1,1,o)== 0 || j(1,1,o)==
255))%window for corner pixels will be 2*2
        %Important - Matlab subtraction does not gives negative answer. Instead it gives 0. So
        carry out subtraction in such a way that the result is positive, i.e. subtract smaller no.
        from larger no.
        % here detail=(j(1,1,o)-avg(1,1,o)) or (avg(1,1,o)-j(1,1,o)) depending on whether the
        pixel has value 255 or 0(in case of noise) respectively so that the subtraction gives a
        positive answer as negative answer is given as 0 by matlab.
        sig(1,1,o)=j(1,1,o);%significant pixels(pixel value(z)=0 or 255)-noise
    else
        insig(1,1,o)=j(1,1,o);%insignificant pixels(pixel value(z)=0 or 255)-image data
    end;
end;
end;

```

Fast Image Restoration by Median Based Lifting Filter using Second Generation

Wavelets

```

start=cputime;
i=imread('lena_gray_salt&pepper10%.tif');

```



```

threshold=5;%Assuming that image data with a value of 0 or 255 will have neighbours
whose values will not vary more than +/-10 from 0 or 255
sizei=size(i);
j=i;
med=i;
%j=-1*ones(sizei);%giving all the pixels the value -1 indicating it is image data
%med=-1*ones(sizei);%giving all the pixels the value -1 indicating it is image data
data=i;
for m = 1:sizei(1,1)
    for n = 1:sizei(1,2)
        if(data(m,n)==0||data(m,n)==255)
            j(m,n)=data(m,n);
        end;
    end;
end;
%j=i(data==0 | data==255);%lifting, storing the value of lifted pixel in j
sig=-1*ones(sizei);%significant pixels(pixel value(z)=0 or 255)-noise
insig=-1*ones(sizei);%insignificant pixels(pixel value(z)=0 or 255)-image data
%Top Left Corner Pixel
med(1,1)=uint8(median([j(1,2) j(2,1) j(2,2)]));
if(((j(1,1)-med(1,1))>threshold || (med(1,1)-j(1,1))>threshold)&&(j(1,1)== 0 || j(1,1)==
255))%window for corner pixels will be 2*2
%Important - Matlab subtraction does not gives negative answer. Instead it gives 0. So
carry out subtraction in such a way that the result is positive, i.e. subtract smaller no.
from larger no.
% here detail=(j(1,1)-med(1,1)) or (med(1,1)-j(1,1)) depending on whether the pixel has
value 255 or 0(in case of noise) respectively so that the subtraction gives a positive
answer as negative answer is given as 0 by matlab.
sig(1,1)=j(1,1);%significant pixels(pixel value(z)=0 or 255)-noise
else
insig(1,1)=j(1,1);%insignificant pixels(pixel value(z)=0 or 255)-image data

```


end;

An Evolutionary Approach to Image Noise Cancellation Utilizing Diminishing Population Particle Swarm Optimization (DP-PSO)

```
i=imread('lena_gray_salt&pepper5%.tif');
sizei=size(i);
j=i;
avg=uint8(0); %Dummy Value for Initialization. '0' has no significance
psoiterations=5; % No of PSO iterations
c1=1;
c2=1;
r1=1; % Can generate this randomly
r2=1; % Can generate this randomly
w=1;
temp1=uint8(0);
temp2=uint8(0);
indices=-1*ones(3,3); % -1 Indicates that this index points to a pixel which is not a
swarm
particle
for m = 1:3:sizei(1,1)
    total=uint16(0);
    cntr=uint8(0);
    for n = 1:3:sizei(1,2)
        if(m==(sizei(1,1)-1)) && (n==(sizei(1,2)-1))
            temp1=m-1;
            temp2=n;
        elseif (n==(sizei(1,2)-1))&&(m==(sizei(1,1)-1))
            temp2=n-1;
            temp1=m;
        elseif (m==(sizei(1,1)-1)) && (n==(sizei(1,2)-1))
```



```

        temp1=m-1;
        temp2=n-1;
    else
        temp1=m;
        temp2=n;
    end;
    for p = 0:2
        for q=0:2
            if((i(temp1+p,temp2+q)~=0)&&(i(temp1+p,temp2+q)~=255))
                cntr=cntr+1;
                total=total+uint16(i(temp1+p,temp2+q));
            end;
        end;
    end;
    avg=uint8(total/uint16(cntr)); %Filter
    total=uint16(0);
    cntr=uint8(0);
    % PSO
    % Initialization of Position and Velocity
    % Particles of the swarm are those pixels which have noise.

```

Improved Evolutionary Approach to Image Noise Cancellation Utilizing Diminishing Population Particle Swarm Optimization (DP-PSO)

```

i=imread('lena_gray_salt&pepper5%.tif');
sizei=size(i);
j=i;
avg=uint8(0); %Dummy Value for Initialization. '0' has no significance
psoiterations=5; % No of PSO iterations
c1=1;
c2=1;
r1=1; % Can generate this randomly

```



```

r2=1; % Can generate this randomly
w=1;
temp1=uint8(0);
temp2=uint8(0);
indices=-1*ones(3,3); % -1 Indicates that this index points to a pixel which is not a
swarm particle
for m = 1:3:sizei(1,1)
    total=uint16(0);
    cntr=uint8(0);
    for n = 1:3:sizei(1,2)
        if(m==(sizei(1,1)-1)) && (n==(sizei(1,2)-1))
            temp1=m-1;
            temp2=n;
        elseif (n==(sizei(1,2)-1))&&(m==(sizei(1,1)-1))
            temp2=n-1;
            temp1=m;
        elseif (m==(sizei(1,1)-1)) && (n==(sizei(1,2)-1))
            temp1=m-1;
            temp2=n-1;
        else
            temp1=m;
            temp2=n;
        end;
    end;

    for p = 0:2
        for q=0:2
            if((i(temp1+p,temp2+q)~=0)&&(i(temp1+p,temp2+q)~=255))
                cntr=cntr+1;
                total=total+uint16(i(temp1+p,temp2+q));
            end;
        end;
    end;
end;

```



```

end;
end;

if(cnr~=0)
avg=uint8(total/uint16(cnr)); %Filter
end;
total=uint16(0);
cnr=uint8(0);
% PSO
% Initialization of Position and Velocity
% Particles of the swarm are those pixels which have noise.

```

CNN Line Detector

```

global A B Bu I m n;
global Im1;

axes(handles.axes2)
if (isrgb(Im1))
Im1=rgb2gray(Im1);
end
imshow(Im1);

Ta=zeros(3,3);
Ta(1,1)=str2double(get(handles.a1,'String'));
Ta(1,2)=str2double(get(handles.a2,'String'));
Ta(1,3)=str2double(get(handles.a3,'String'));
Ta(2,1)=str2double(get(handles.a4,'String'));
Ta(2,2)=str2double(get(handles.a5,'String'));
Ta(2,3)=str2double(get(handles.a6,'String'));
Ta(3,1)=str2double(get(handles.a7,'String'));

```



```
Ta(3,2)=str2double(get(handles.a8,'String'));
Ta(3,3)=str2double(get(handles.a9,'String'));
```

```
Tb=zeros(3,3);
Tb(1,1)=str2double(get(handles.b1,'String'));
Tb(1,2)=str2double(get(handles.b2,'String'));
Tb(1,3)=str2double(get(handles.b3,'String'));
Tb(2,1)=str2double(get(handles.b4,'String'));
Tb(2,2)=str2double(get(handles.b5,'String'));
Tb(2,3)=str2double(get(handles.b6,'String'));
Tb(3,1)=str2double(get(handles.b7,'String'));
Tb(3,2)=str2double(get(handles.b8,'String'));
Tb(3,3)=str2double(get(handles.b9,'String'));
```

```
I=str2double(get(handles.bias,'String'));
```

```
A=Ta
```

```
B=Tb
```

```
u = im2double(Im1);
```

```
uu = im2double(max(max(u)));
```

```
ul = im2double(min(min(u)));
```

```
u = (u-ul)/(uu-ul)*2-1;
```

```
x0 = u;
```

```
colormap(gray(64))
```

```
t = 0;
```

```
Xt = x0;
```

```
tf = str2double(get(handles.time1,'String'));
```

```
dtime = str2double(get(handles.dt1,'String'));
```



```

m=0;
n=0;
global Im2;

%----- Start Calculation CNN -----
while(t<tf)
    Im2=image((pwlsg(Xt)+1)*32);
    %axis('image');
    drawnow
    tnext = min([tf,t+dt]);

    Atem = A;
    Bu = conv2(u,B,'same');
    [m,n] = size(x0);
    x0 = x0(:);
    [t,y] = ode23('cnnderiv', [0, tf/2, tf], x0);
    ly = size(y,1);
    x0 = reshape(y(ly,:),m,n);
    Xt=x0;

    t = tnext
    set(handles.showtime1,'String',num2str(t));
    %pause(0.01);

end;

```

Project Main Interface

```

function varargout = BTechProject_Tushar_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure

```



```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
```

```
varargout{1} = handles.output;
```

```
% -----
```

```
function FileMenu_Callback(hObject, eventdata, handles)
```

```
% hObject handle to FileMenu (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
% -----
```

```
function OpenMenuItem_Callback(hObject, eventdata, handles)
```

```
% hObject handle to OpenMenuItem (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
file = uigetfile('*.fig');
```

```
if ~isequal(file, 0)
```

```
    open(file);
```

```
end
```

```
% -----
```

```
function PrintMenuItem_Callback(hObject, eventdata, handles)
```

```
% hObject handle to PrintMenuItem (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
printdlg(handles.figure1)
```



```

% -----
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to CloseMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
    ['Close ' get(handles.figure1,'Name') '...'],...
    'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)

```

```

% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```


% Hints: contents = get(hObject,'String') returns listbox1 contents as cell array

% contents {get(hObject,'Value')} returns selected item from listbox1

% --- Executes during object creation, after setting all properties.

function listbox1_CreateFcn(hObject, eventdata, handles)

% hObject handle to listbox1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),

get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in pushbutton1.

function pushbutton1_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

val = get(handles.listbox1,'Value');

str = get(handles.listbox1,'String');

switch str{val}

case 'LFSGW'

lfsgw_gui

case 'Improved DP-PSO'


```
    dppso_gui
case 'Median based Lifting Filter'
    medlf_gui
case 'CNN'
    gui_test
case 'MSE and PSNR'
    mse_psnr_gui
otherwise
    disp('No Selection has been made.')
end
```