**FUSION BASED BRAIN TUMOR DETECTION USING MACHINE LEARTNING**

**ABSTRACT**

Medical Image fusion plays a vital role in medical field to diagnose the brain tumors which can be classified as benign or malignant. It is the process of integrating multiple images of the same scene into a single fused image to reduce uncertainty and minimizing redundancy while extracting all the useful information from the source images. SVM is used to fuse two brain MRI images with different vision. The fused image will be more informative than the source images. The texture and wavelet features are extracted from the fused image. The SVM Classifier classifies the brain tumors based on trained and tested features. The proposed method achieved 80.48% sensitivity, 99.9% specificity and 99.69% accuracy. Experimental results obtained from fusion process prove that the use of the proposed image fusion approach shows better performance while compared with conventional fusion methodologies.

**INTRODUCTION**

Medical image processing has developed as one of the critical factors in regular clinical applications, such as disease diagnosis and treatment planning. Owing to the technical limitations, the quality of medical images is usually unsatisfactory, degrading the accuracy of human interpretation and further medical image analysis, thereby, requiring the quality of these images to be enhanced. One approach to enhance the image quality is by image denoising. Several denoising approaches, like adaptive filters, wavelet-based methods, etc. were proposed. Another efficient technique is by image fusion which improves the image quality by combining the corresponding information from multimodal images into a single fused image. This resulting image is called as fused image.A fusion process is nothing but a combination of salient information in order to synthesize an image with more information than individual image and synthesized image is more suitable for visual perception. Image fusion is a process of combining multiple input images of the same scene into a single fused image, which preserves full content information and also retaining the important features from each of the original images. The fused image should have more useful information content compared to the individual image. Radiologists mostly prefer both MR and CT images side by side, when both images are available. This provides them all the available image information, but its accessibility is limited to visual correlation between the two images. Both CT and MR images can be employed as it is difficult to determine whether narrowing of a spinal canal is caused by a tissue or bone. Both the CT and MR modalities provide complementary information. In order to properly visualize the related bone and soft tissue structures, the images must be mentally aligned and fused together. This process leads to more accurate data interpretation and utility. In fundamental multi-modal image fusion methodologies, the source image is just overlaid by assigning them to different color channels. In color image fusion, this overlay approach is used to expand the amount of information over a single image, but it does not affect the image contrast or distinguish the image features. So in this paper we propose a novel region based image fusion algorithm for multifocus and multimodal images which also overcomes the limitations of different approaches.

**EXSITING METHODS:**

* Image fusion technique based on DWT(Discrete Wavelet Transform).
* Hyper spectral Image fusion based on PCT(Principal Components Transform).
* Fusion technique based on NSCT(Non-Sub sampled Contourlet Transform).

**PROPOSED METHOD:**

In our proposed system, Medical image fusion combines different modality of medical images to produce a high quality fused image with spatial and spectral information. The fused image with more information improved the performance of image analysis algorithms used in different medical diagnosis applications.SVM is used in this paper for brain image fusion and K-Clusterin features are extracted from the fused brain image. The brain tumor region is segmented using the extracted features and SVM classifier helps to identify whether the tumor is benign or malignant. Thus it helps the physician and radiologist for brain tumor diagnosis for human surgery.

**Literature survey:**

1. **BRAIN TUMOUR DETECTION USING HOG BY SVM**

**Year-2017**

**Praveena Pedapati ,Rama Vaishnavi Tanneedi**

Detection of a brain tumour in medical images is always a challenging task. Factors like size, shape, and position of tumour vary from different patient’s brain. So, it's important to know the exact shape, size and position of a tumour in the brain making it a challenging task for detection. Some patients exhibit high glioma (HG) type tumor while others show low glioma (LG) type. So, knowing the detailed properties of a tumour to detect them in medical images is mandatory. So far many algorithms have been implemented on how to detect and extract the tumours in medical images, they used techniques such as hybrid approach with Support Vector Machine (SVM), back propagation and dice coefficient. Among these algorithm which used back propagation as base classifier had a highest accuracy of 90%. In this work feature extraction of the medical images of patients’ tumors in database is extracted using Histogram of Oriented Gradient, later these images are classified into tumor and non tumor images using SVM. The detection of brain tumours in patient’s image is achieved by testing the performance of SVM based on Receiver Operating Characteristics (ROC). ROC include true positive rate, true negative rate, false positive rate and false negative rate. Using ROC we calculated accuracy, sensitivity and specificity values for all the medical images of the database. For image data folder of HG in vector form, SVM gave an accuracy of 97% for 95th slice of T1 modality with high true positive rate of 0.97 remaining highest among other modalities. Whereas SVM gave an accuracy of 87% for 135th slice of T1 modality with high true positive rate of 0.8 and low false positive rate of 0.06 among other image data folder of HG. For image data folder of LG, SVM gave an accuracy of 62% for the 90th slice of FLAIR modality with the high true positive rate of 0.5 and low false positive rate of 0.25 among all others. For synthetic data folder of HG, SVM gave an accuracy of 62% for a 100th slice of FLAIR modality with the high true positive rate of 0.5 and low false positive rate of 0.06 among all others. For synthetic data folder of LG, SVM gave an accuracy of 62% for a 100th slice of FLAIR modality with the high true positive rate of 0.5 and low false positive rate of 0.06 among all others.

1. **Detecting Brain Tumour using K-Mean Clustering and Morphological Operations**

**Year-2018**

Shaheen M. Khan, Radhika S. Kharade

Brain tumour is inherently serious and life-threatening problem because of its character in the limited space of the intracranial cavity (space formed inside the skull). Tumour is the one of the most common brain disease and this is the reason for the diagnosis & treatment of the brain tumour has vital importance. CT scan is the technique used to produce computerized image of internal body tissues. Cells are growing in uncontrollable manner this leads to mass of unwanted tissue that is termed as neoplasm. Normally the anatomy of the Brain can be viewed by the CT scan. It is not affect the human body. Because it doesn’t use any radiation. In this paper we proposed segmentation of brain CT Scan Image using K-means clustering algorithm followed by morphological filtering which avoids the misclustered regions that can be formed after segmentation of the brain CT Scan Image for detection of tumour location.

1. **Comparison of Feature Selection Techniques for Detection of Malignant Tumor in Brain Images**

**Year-2006** Sasikala Mohan Kumaravel Natesan

This paper presents and compares feature selection algorithms for the detection of glioblastoma multiforme in brain images. Texture features are extracted from normal and tumor regions (ROI) using spatial grey level dependence method and wavelet transform. An artificial neural network has been used for classification. A very difficult problem in classification. Techniques is the choice of features to distinguish between classes. The feature optimization problem is addressed using a genetic algorithm (GA) as a search method. Principal component analysis, classical sequential methods and floating search algorithm are compared against the genetic approach in terms of the best recognition rate achieved and the optimal number of features. The classification performance of 97.3% is achieved in GA with optimal features compared to sequential methods and Principal component analysis.

1. **Image Classification using Support Vector Machine and Artificial Neural Network**

Year- 2012 Le Hoang Thai

Image classification is one of classical problems of concern in image processing. There are various approaches for solving this problem. The aim of this paper is bring together two areas in which are Artificial Neural Network (ANN) and Support Vector Machine (SVM) applying for image classification. Firstly, we separate the image into many sub-images based on the features of images. Each sub-image is classified into the responsive class by an ANN. Finally, SVM has been compiled all the classify result of ANN. Our proposal classification model has brought together many ANN and one SVM. Let it denote ANN\_SVM. ANN\_SVM has been applied for Roman numerals recognition application and the precision rate is 86%. The experimental results show the feasibility of our proposal model.

1. **Brain Tumor Segmentation With Symmetric Texture And symmetric Intensity Based Decision Forests**

**Year-2013** Anthony Bianchi ,James V. Miller

Accurate automated segmentation of brain tumors in MR images is challenging due to overlapping tissue intensity distributions and amorphous tumor shape. However, a clinically viable solution providing precise quantification of tumor and edema volume would enable better pre-operative planning, treatment monitoring and drug development. Our contributions are threefold. First, we design efficient gradient and LBPTOP based texture features which improve classification accuracy over standard intensity features. Second, we extend our texture and intensity features to symmetric texture and symmetric intensity which further improve the accuracy for all tissue classes. Third, we demonstrate further accuracy enhancement by extending our long range features from 100 mm to a full 200 mm. We assess our brain segmentation technique on 20 patients in the BraTS 2012 dataset. Impact from each contribution is measured and the combination of all the features is shown to yield state-of-the-art accuracy and speed.

1. "**Segmentation of Brain Tumor Images Based on Integrated Hierarchical Classification and Regularization**

Year-2012

Stefan Bauer , Thomas Fejes

We propose a fully automatic method for brain tumor segmentation, which integrates random forest classification with hierarchical conditional random field regularization in an energy minimization scheme. It has been evaluated on the BRATS2012 dataset, which contains low- and high-grade gliomas from simulated and real-patient images. The method achieved convincing results (average Dice coefficient: 0.73 and 0.59 for tumor and edema respectively) within a reasonably fast computation time (approximately 4 to 12 minutes).

1. **Automatic Detection of Brain Tumor Using KMeans Clustering**

**Year- 2017**

**Nitesh Kumar Singh , Geeta Singh**

Brain tumor is an uncommon and uncontrolled growth of cell in brain. In medical image processing one of the most challenging tasks is study of Brain tumor. Recently, magnetic resonance imaging (MRI) is the most widely used imaging modality for identifying brain tumor and other discrepancies. From these MRI images, we are able to determine the detailed anatomical information to examine the development of the human brain and to diagnose the various diseases. The tumor detection becomes most complicated for the huge image database. So a software approach is required to help the accurate and faster clinical diagnosis. The proposed system identifies and segments the tumor portions of the image successfully using MATLAB. The aim is to introduce an algorithm employed for performing useful operations on the MRI images such as filtering, pre-processing, morphological operation, K-means clustering, feature Extraction, Decision making system, SVM Classifier & Naïve Bayes. This work helps to save the time of both pathologist and doctor. The experimental results show that the proposed method detects tumor areas efficiently in the image of the brain.

1. **Automatic detection and severity analysis of brain tumors using gui in matlab**

**Year- 2013**

[**Mallampalli Srilakshmi Karuna**](https://www.semanticscholar.org/author/Mallampalli-Srilakshmi-Karuna/34681672)**,** [**Ankita Joshi**](https://www.semanticscholar.org/author/Ankita-Joshi/1888091)

Medical image processing is the most challenging and emerging field now a day’s processing of MRI images is one of the parts of this field. The quantitative analysis of MRI brain tumor allows obtaining useful key indicators of disease progression. This is a computer aided diagnosis systems for detecting malignant texture in biological study. This paper presents an approach in computer-aided diagnosis for early prediction of brain cancer using Texture features and neuro classification logic. This paper describes the proposed strategy for detection; extraction and classification of brain tumour from MRI scan images of brain; which incorporates segmentation and morphological functions which are the basic functions of image processing. Here we detect the tumour, segment the tumour and we calculate the area of the tumour. Severity of the disease can be known, through classes of brain tumour which is done through neuro fuzzy classifier and creating a user friendly environment using GUI in MATLAB. In this paper cases of 10 patients is taken and severity of disease is shown and different features of images are calculated.

1. **Brain Tumor volume measurement: comparison of manual and semiautomated methods.**

**Year- 1999**

[**Bonnie N. Joe**](https://www.semanticscholar.org/author/Bonnie-N.-Joe/11033280)**,** [**Melanie B. Fukui**](https://www.semanticscholar.org/author/Melanie-B.-Fukui/5687626)**,** [**Carolyn C. Meltzer**](https://www.semanticscholar.org/author/Carolyn-C.-Meltzer/30885893)

To compare the reliability of two approaches to measuring enhancing brain tumor volumes--the conventional manual trace method and a threshold-based, semi automated computer software method. Two operators rated contrast material-enhanced, T1-weighted axial magnetic resonance (MR) image data sets from 16 patients aged 21-71 years with high-grade gliomas. Each MR data set was rated twice by using manual tracing and twice by using the semiautomated method. The semiautomated measurement method involved a thresholding algorithm based on mixture modeling. The data collection time for each method was recorded. Reliability was measured by using inter- and intraoperator agreement indexes. The semiautomated computer method of measuring tumor volume was faster than the manual trace method. Semiautomated computer approaches offer an alternative to manual tracing for measuring serial tumor volumes in patients with high-grade brain neoplasms.

1. **Automated detection and quantification of residual brain tumor using an interactive computer-aided detection scheme**

**Year-2017**

**Kevin P. Gaffney, Faranak Aghaei, James Battiste, Bin Zheng**

Detection of residual brain tumor is important to evaluate efficacy of brain cancer surgery, determine optimal strategy of further radiation therapy if needed, and assess ultimate prognosis of the patients. Brain MR is a commonly used imaging modality for this task. In order to distinguish between residual tumor and surgery induced scar tissues, two sets of MRI scans are conducted preand post-gadolinium contrast injection. The residual tumors are only enhanced in the post-contrast injection images. However, subjective reading and quantifying this type of brain MR images faces difficulty in detecting real residual tumor regions and measuring total volume of the residual tumor. In order to help solve this clinical difficulty, we developed and tested a new interactive computer-aided detection scheme, which consists of three consecutive image processing steps namely, 1) segmentation of the intracranial region, 2) image registration and subtraction, 3) tumor segmentation and refinement. The scheme also includes a specially designed and implemented graphical user interface (GUI) platform. When using this scheme, two sets of preand post-contrast injection images are first automatically processed to detect and quantify residual tumor volume. Then, a user can visually examine segmentation results and conveniently guide the scheme to correct any detection or segmentation errors if needed. The scheme has been repeatedly tested using five cases. Due to the observed high performance and robustness of the testing results, the scheme is currently ready for conducting clinical studies and helping clinicians investigate the association between this quantitative image marker and outcome of patients.

1. **Robust kernelized local information fuzzy C-means clustering for brain magnetic resonance image segmentation.**

**Year-2016**

**Ahmed Elazab, Yousry M Abdulazeem, Shiqian Wu, Qingmao Hu**

Brain tissue segmentation from magnetic resonance (MR) images is an importance task for clinical use. The segmentation process becomes more challenging in the presence of noise, grayscale inhomogeneity, and other image artifacts. In this paper, we propose a robust kernelized local information fuzzy C-means clustering algorithm (RKLIFCM). It incorporates local information into the segmentation process (both grayscale and spatial) for more homogeneous segmentation. In addition, the Gaussian radial basis kernel function is adopted as a distance metric to replace the standard Euclidean distance. The main advantages of the new algorithm are: efficient utilization of local grayscale and spatial information, robustness to noise, ability to preserve image details, free from any parameter initialization, and with high speed as it runs on image histogram. We compared the proposed algorithm with 7 soft clustering algorithms that run on both image histogram and image pixels to segment brain MR images. Experimental results demonstrate that the proposed RKLIFCM algorithm is able to overcome the influence of noise and achieve higher segmentation accuracy with low computational complexity.

**SYSTEM REQUIREMENTS:**

**GENERAL:**

The system requirement of the project is described and the specification of the software and hardware requirements of the project is described.

**HARDWARE REQUIREMENTS:**

* Processor Type : Pentium -IV
* Speed : 2.4 GHZ
* Ram : 128 MB RAM
* Hard disk : 20 GB HD

**SOFTWARE REQUIREMENTS**

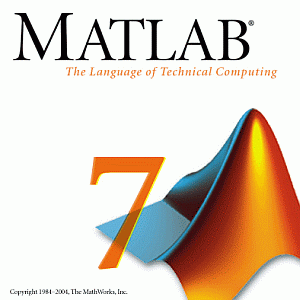
* Operating System : Windows 7
* Software Programming Package: Matlab R2014a

**INTRODUCTION TO MATLAB:**

**MATLAB VERSION REQUIRED- 2018a**

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

* Math and computation
* Algorithm development
* Data acquisition
* Modeling, simulation, and prototyping
* Data analysis, exploration, and visualization
* Scientific and engineering graphics
* Application development, including graphical user interface building



**Fig**

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non interactive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most uses of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M – files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

**1.7.1 The MATLAB system:**

The MATLAB system consists of five main parts

* **Development Environment**:

This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and command window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.

* **The MATLAB Mathematical Function Library**:

This is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix Eigen values, Bessel functions, and fast Fourier transforms.

* **The MATLAB Language**:

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both “programming in the small” to rapidly create quick and dirty throw-away programs, and “programming in the large” to create large and complex application programs.

* **Graphics:**

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization,

image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

* **The MATLAB Application Program Interface (API):**

This is a library that allows you to write C and FORTRAN programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

Various toolboxes are there in MATLAB for computing recognition techniques, but we are using **IMAGE PROCESSING** toolbox.

**SOFTWARE DESCRIPTION**

**Getting Started**

If you are new to MATLAB, you should start by reading Manipulating Matrices. The most important things to learn are how to enter matrices, how to use the: (colon) operator, and how to invoke functions. After you master the basics, you should read the rest of the sections below and run the demos.

At the heart of MATLAB is a new language you must learn before you can fully exploit its power. You can learn the basics of MATLAB quickly, and mastery comes shortly after. You will be rewarded with high productivity, high-creativity computing power that will change the way you work.

**Introduction** - describes the components of the MATLAB system.

**Development Environment** - introduces the MATLAB development environment, including information about tools and the MATLAB desktop.

**Manipulating Matrices** - introduces how to use MATLAB to generate matrices and perform mathematical operations on matrices.

**Graphics** - introduces MATLAB graphic capabilities, including information about plotting data, annotating graphs, and working with images.

**Programming with MATLAB** - describes how to use the MATLAB language to create scripts and functions, and manipulate data structures, such as cell arrays and multidimensional arrays.

**INTRODUCTION**

What Is MATLAB?

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

* Math and computation
* Algorithm development
* Modeling, simulation, and prototyping
* Data analysis, exploration, and visualization
* Scientific and engineering graphics
* Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB uses software developed by the LAPACK and ARPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

**The MATLAB System**

The MATLAB system consists of five main parts:

**Development Environment**. This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, and browsers for viewing help, the workspace, files, and the search path.

**The MATLAB Mathematical Function Library**. This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

**The MATLAB Language**. This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

**Handle Graphics®**. This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

**The MATLAB Application Program Interface (API).** This is a library that allows you to write C and FORTRAN programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

**DEVELOPMENT ENVIRONMENT**

**Introduction**

This chapter provides a brief introduction to starting and quitting MATLAB, and the tools and functions that help you to work with MATLAB variables and files. For more information about the topics covered here, see the corresponding topics under Development Environment in the MATLAB documentation, which is available online as well as in print.

**Starting and Quitting MATLAB**

**Starting MATLAB**

On a Microsoft Windows platform, to start MATLAB, double-click the MATLAB shortcut icon on your Windows desktop.

On a UNIX platform, to start MATLAB, type matlab at the operating system prompt.

After starting MATLAB, the MATLAB desktop opens - see MATLAB Desktop.

You can change the directory in which MATLAB starts, define startup options including running a script upon startup, and reduce startup time in some situations.

**Quitting MATLAB**

To end your MATLAB session, select Exit MATLAB from the File menu in the desktop, or type quit in the Command Window. To execute specified functions each time MATLAB quits, such as saving the workspace, you can create and run a finish.m script.

**MATLAB Desktop**

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

The first time MATLAB starts, the desktop appears as shown in the following illustration, although your Launch Pad may contain different entries.

You can change the way your desktop looks by opening, closing, moving, and resizing the tools in it. You can also move tools outside of the desktop or return them back inside the desktop (docking). All the desktop tools provide common features such as context menus and keyboard shortcuts.

You can specify certain characteristics for the desktop tools by selecting Preferences from the File menu. For example, you can specify the font characteristics for Command Window text. For more information, click the Help button in the Preferences dialog box.

**Desktop Tools**

This section provides an introduction to MATLAB's desktop tools. You can also use MATLAB functions to perform most of the features found in the desktop tools. The tools are:

* Current Directory Browser
* Workspace Browser
* Array Editor
* Editor/Debugger
* Command Window
* Command History
* Launch Pad
* Help Browser

**Command Window**

Use the Command Window to enter variables and run

functions and M-files.

**Command History**

Lines you enter in the Command Window are logged in the Command History window. In the Command History, you can view previously used functions, and copy and execute selected lines. To save the input and output from a MATLAB session to a file, use the diary function.

**Running External Programs**

You can run external programs from the MATLAB Command Window. The exclamation point character! is a shell escape and indicates that the rest of the input line is a command to the operating system. This is useful for invoking utilities or running other programs without quitting MATLAB. On Linux, for example,!emacs magik.m invokes an editor called emacs for a file named magik.m. When you quit the external program, the operating system returns control to MATLAB.

**Launch Pad**

MATLAB's Launch Pad provides easy access to tools, demos, and documentation.

**Help Browser**

Use the Help browser to search and view documentation for all your Math Works products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents.

To open the Help browser, click the help button in the toolbar, or type helpbrowser in the Command Window. The Help browser consists of two panes, the Help Navigator, which you use to find information, and the display pane, where you view the information.

**Help Navigator**

Use to Help Navigator to find information. It includes:

**Product filter** - Set the filter to show documentation only for the products you specify.

**Contents tab** - View the titles and tables of contents of documentation for your products.

**Index tab** - Find specific index entries (selected keywords) in the MathWorks documentation for your products.

**Search tab** - Look for a specific phrase in the documentation. To get help for a specific function, set the Search type to Function Name.

**Favorites tab** - View a list of documents you previously designated as favorites.

**Display Pane**

After finding documentation using the Help Navigator, view it in the display pane. While viewing the documentation, you can:

**Browse to other pages** - Use the arrows at the tops and bottoms of the pages, or use the back and forward buttons in the toolbar.

**Bookmark pages** - Click the Add to Favorites button in the toolbar.

**Print pages** - Click the print button in the toolbar.

**Find a term in the page** - Type a term in the Find in page field in the toolbar and click Go.

Other features available in the display pane are: copying information, evaluating a selection, and viewing Web pages.

**Current Directory Browser**

MATLAB file operations use the current directory and the search path as reference points. Any file you want to run must either be in the current directory or on the search path.

**Search Path**

To determine how to execute functions you call, MATLAB uses a search path to find M-files and other MATLAB-related files, which are organized in directories on your file system. Any file you want to run in MATLAB must reside in the current directory or in a directory that is on the search path. By default, the files supplied with MATLAB and MathWorks toolboxes are included in the search path.

**Workspace Browser**

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces.

To view the workspace and information about each variable, use the Workspace browser, or use the functions who and whos.

To delete variables from the workspace, select the variable and select Delete from the Edit menu. Alternatively, use the clear function.

The workspace is not maintained after you end the MATLAB session. To save the workspace to a file that can be read during a later MATLAB session, select Save Workspace As from the File menu, or use the save function. This saves the workspace to a binary file called a MAT-file, which has a .mat extension. There are options for saving to different formats. To read in a MAT-file, select Import Data from the File menu, or use the load function.

**Array Editor**

Double-click on a variable in the Workspace browser to see it in the Array Editor. Use the Array Editor to view and edit a visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace.

**Editor/Debugger**

Use the Editor/Debugger to create and debug M-files, which are programs you write to runMATLAB functions. The Editor/Debugger provides a graphical user interface for basic textediting, as well as for M-file debugging.

You can use any text editor to create M-files, such as Emacs, and can use preferences (accessible from the desktop File menu) to specify that editor as the default. If you use another editor, you can still use the MATLAB Editor/Debugger for debugging, or you can use debugging functions, such as dbstop, which sets a breakpoint.

If you just need to view the contents of an M-file, you can display it in the Command Window by using the type function.

**MANIPULATING MATRICES**

**Entering Matrices**

The best way for you to get started with MATLAB is to learn how to handle matrices. Start MATLAB and follow along with each example.

You can enter matrices into MATLAB in several different ways:

* Enter an explicit list of elements.
* Load matrices from external data files.
* Generate matrices using built-in functions.
* Create matrices with your own functions in M-files.

Start by entering Dürer's matrix as a list of its elements. You have only to follow a few basic conventions:

* Separate the elements of a row with blanks or commas.
* Use a semicolon, ; , to indicate the end of each row.
* Surround the entire list of elements with square brackets, [ ].

To enter Dürer's matrix, simply type in the Command Window

A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]

MATLAB displays the matrix you just entered.

A =

16 3 2 13

5 10 11 8

9 6 7 12

4 15 14 1

This exactly matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A.

**Expressions**

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are:

* Variables
* Numbers
* Operators
* Functions

**Variables**

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

num\_students = 25

Creates a 1-by-1 matrix named num\_students and stores the value 25 in its single element.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are not the same variable. To view the matrix assigned to any variable, simply enter the variable name.

**Numbers**

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter e to specify a power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of legal numbers are

3 -99 0.0001

9.6397238 1.60210e-20 6.02252e23

1i -3.14159j 3e5i

All numbers are stored internally using the long format specified by the IEEE floating-point standard. Floating-point numbers have a finite precision of roughly 16 significant decimal digits and a finite range of roughly 10-308 to 10+308.

**Operators**

Expressions use familiar arithmetic operators and precedence rules.

|  |  |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| \ | Left division (described in "Matrices and Linear Algebra" in Using MATLAB) |
| ^ | Power |
| ' | Complex conjugate transpose |
| ( ) | Specify evaluation order |

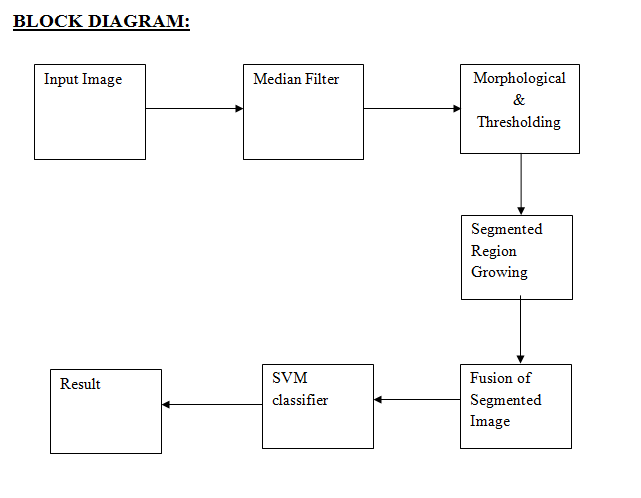
**Functions**

MATLAB provides a large number of standard elementary mathematical functions, including abs, sqrt, exp, and sin. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically. MATLAB also provides many more advanced mathematical functions, including Bessel and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

help elfun

Some of the functions, like sqrt and sin, are built-in. They are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other functions, like gamma and sinh, are implemented in M-files. You can see the code and even modify it if you want. Several special functions provide values of useful constants.

|  |  |
| --- | --- |
| Pi | 3.14159265... |
| i | Imaginary unit, √-1 |
| I | Same as i |
| Eps | Floating-point relative precision, 2-52 |
| Realmin | Smallest floating-point number, 2-1022 |
| Realmax | Largest floating-point number, (2- **ε**)21023 |
| Inf | Infinity |
| NaN | Not-a-number |

****

**ER DIAGRAM**

RESULT

ORIGINAL IMAGE

SVM CLASSIFIER

SEGMENTATION

THRESHOLDING

**SEQUENCE DIAGRAM**

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

****

**USE CASE:**

A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components a user or another system that will interact with the system modelled. A use case is an external view of the system that represents some action the user might perform in order to complete a task.

****

**Collaborative:**

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time.

****

**ALGORITHM DISCRIPTION**

**MEDIAN FILTERING**

* Median filter- Median filter is used for denoising the image
* This is a important step for image Enhancement
* Noise reduction is a typical pre-processing step to improve the results of later processing.
* It preserves edges while removing noise.
* The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the median of neighbouring entries.

**K MEANS CLUSTERING**

* k-means is  one of  the simplest unsupervised  learning  algorithms  that  solve  the well  known clustering problem.
* The procedure follows a simple and  easy  way  to classify a given data set  through a certain number of  clusters
* The  main  idea  is to define k centres, one for each cluster.
* These centres  should  be placed in a cunning  way  because of  different  location  causes different  result.
* So, the better  choice  is  to place them  as  much as possible  far away from each other.
* The  next  step is to take each point belonging  to a  given data set and associate it to the nearest centre.

**SVM ALGORITHM**

* “Support Vector Machine” (SVM) is a supervised machine learning algorithm
* which can be used for both classification or regression challenges.
* In this algorithm, we plot each data item as a point in n-dimensional space
* where n is number of features with the value of each feature being the value of a particular coordinate.
* Then, we perform classification by finding the hyper-plane that differentiate the two classes very well

**METHODOLOGY**

* **PREPROCESSING**-Preprocessing the input image
* **IMAGE ENCHANCEMENT** -by denoising the image using the algorithm called Median Filter
* **FEARTURES EXTRACTION**-Extracting the morphological features by using the k –means clustering algorithm
* **THRESHOLDING-**In addition to thresholding the extracting image
* Thresholding is the simplest method of image segmentation. From a grayscale image, thresholding can be used to create binary images
* **SEGMENTATION**-Then segmentation process will carried out for further performance to identified the tumor is benign and malignant
* **SVM-**Now the Support Vector Machine(SVM) classifier is used for classification as well as in regression condition
* SVM Classifies the tumor is benign or malignant

**SYSTEM TESTING**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**TYPES OF TESTS**

**Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

***Integration testing***

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

**Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**6.1 Unit Testing:**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

**Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

# 6.2 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

***6.3 Acceptance Testing***

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Coding:**

**Main coding:**

clc;

close all;

clear all;

% To read images from file..

[filename, pathname] = uigetfile('\*.\*','Load Image 1 ');cd(pathname);

im1=imread([pathname,filename]);

[filename, pathname] = uigetfile('\*.\*','Load Image 2 ');cd(pathname);

im2=imread([pathname,filename]);

figure(1);

subplot(121);imshow(im1,[]);

subplot(122);imshow(im2,[]);

% image decomposition using discrete stationary wavelet transform

[A1L1,H1L1,V1L1,D1L1] = swt2(im1,1,'sym2');

[A2L1,H2L1,V2L1,D2L1] = swt2(im2,1,'sym2');

% fusion start

AfL1 = 0.5\*(A1L1+A2L1);

D = (abs(H1L1)-abs(H2L1))>=0;

HfL1 = D.\*H1L1 + (~D).\*H2L1;

D = (abs(V1L1)-abs(V2L1))>=0;

VfL1 = D.\*V1L1 + (~D).\*V2L1;

D = (abs(D1L1)-abs(D2L1))>=0;

DfL1 = D.\*D1L1 + (~D).\*D2L1;

% Fused Image

imf = iswt2(AfL1,HfL1,VfL1,DfL1,'sym2');

figure(2); imshow(imf,[]);

% compute PCA

im1 = double(im1);

im2 = double(im2);

C = cov([im1(:) im2(:)]);

[V, D] = eig(C);

if D(1,1) >= D(2,2)

pca = V(:,1)./sum(V(:,1));

else

pca = V(:,2)./sum(V(:,2));

end

% Fused Image

imf = pca(1)\*im1 + pca(2)\*im2;

figure(3); imshow(uint8(imf),[]);

%% Otsu segmentation..

imf = uint8(imf);

level = graythresh(imf);

seg=im2bw(imf,0.7);

figure(4),imshow(double(seg),[]);

%performance analysis for fused image

std\_dev=std2(imf);

disp('standard deviation of fused image=');

disp(std\_dev)

%performance analysis for seg image

std\_dev1=std2(seg);

disp('standard deviation of segmented image=');

disp(std\_dev1)

% [Gmag, Gdir] = imgradient(seg,'prewitt');

%%gradient of fused image

g=imgradient(imf);

g=g(1,:);

grad=mean(g);

disp(' gradient of fused image=');

disp(grad)

%%gradient of segmented image

g1=imgradient(seg);

g1=g1(1,:);

grad1=mean(g1);

disp(' gradient of segmented image=');

disp(grad1)

%%spectral discrepancy of fused image

sd=fft(imf);

sd=sd(1,:);

sd=mean(sd);

sd=real(sd);

disp('spectral discrepancy of fused image=')

disp(sd)

%%spectral discrepancy of segmented image

sd=fft(seg);

sd=sd(1,:);

sd=mean(sd);

sd=real(sd);

disp('spectral discrepancy of segmented image=')

disp(sd)

%% feature extraction

% level = graythresh(c)

% seg=im2bw(c,level)

% figure(4),imshow(seg);

x = double(seg);

m = size(seg,1);

n = size(seg,2);

signal1 = seg(:,:);

%% Haar Feature Extraction..

[cA,cH,cV,cD] = dwt2(seg,'haar');

cA = mean2(cA);

cH = mean2(cH);

cV = mean2(cV);

cD = mean2(cD);

Mean = mean2(seg);

Standard\_Deviation = std2(seg);

Entropy = entropy(seg);

Variance = mean2(var(double(seg)));

a = sum(double(seg(:)));

Smoothness = 1-(1/(1+a));

Kurtosis = kurtosis(double(seg(:)));

Skewness = skewness(double(seg(:)));

% Inverse Difference Movement

m = size(seg,1);

n = size(seg,2);

in\_diff = 0;

for i = 1:m

for j = 1:n

temp = seg(i,j)./(1+(i-j).^2);

in\_diff = in\_diff+temp;

end

end

IDM = double(in\_diff);

featext\_fcm = [Mean, Standard\_Deviation, Entropy,Variance, Smoothness, Kurtosis, Skewness, IDM, cA, cH,cV,cD];

%% Classifier..

load Trainfeat\_fcm.mat

load label.mat

xdata = Trainfeat\_fcm;

group = label;

% svmstruct = svmtrain(xdata,group);

% Result = svmclassify(svmstruct, featext\_fcm);

% xdata = Trainfeat;

% group = label;

svmstruct = fitcsvm(xdata,group,'KernelFunction', 'linear');

[Result,s] = predict(svmstruct, featext\_fcm );

if Result == 1

msgbox('BENGIN');

else

msgbox('MALIGANT');

end

%% Performance Analysis..

% load Result\_fcm.mat

% cp = classperf(label,Result\_fcm);

% accuracy = cp.CorrectRate

% sensitivity = cp.Sensitivity

% specificity = cp.Specificity

% confusionmat = confusionmat(label,Result\_fcm);

**Function coding:**

1. **Pc\_fusion function:**

function Fs = PC\_fusion(a,b)

% The code is written by Kun Zhan, Jicai Teng, Qiaoqiao Li

% $Revision: 1.0.0.0 $ $Date: 2014/03/20 $ â€?â€Ž00:30:25 $

w = ones(8);

a = im2double(a);

b = im2double(b);

Ac = phasecong3(a);

Bc = phasecong3(b);

M = Ac > Bc;

M1 = M - 0.5;

M1 = conv2(M1, w, 'same');

M = conv2(M1, w, 'same');

M = better\_Y(M);

M = kron(M,w);

Fs = M.\*a+(1-M).\*b;

Fs = uint8(double(Fs)\*255);

function Z = better\_Y(Y)

xj = padarray(Y,[3 3],'symmetric');

Z=conv2(double(xj),ones(7),'valid')>0;

function Y = phasecong3(varargin)

[im, nscale, norient, minWaveLength, mult, sigmaOnf, ...

k, cutOff, g, noiseMethod] = checkargs(varargin(:));

epsilon = .0001; % Used to prevent division by zero.

[rows,cols] = size(im);

imagefft = fft2(im); % Fourier transform of image

zero = zeros(rows,cols);

EO = cell(nscale, norient); % Array of convolution results.

PC = cell(norient,1);

covx2 = zero; % Matrices for covariance data

covy2 = zero;

covxy = zero;

EnergyV = zeros(rows,cols,3); % Matrix for accumulating total energy

% vector, used for feature orientation

% and type calculation

pcSum = zeros(rows,cols);

% Pre-compute some stuff to speed up filter construction

% Set up X and Y matrices with ranges normalised to +/- 0.5

% The following code adjusts things appropriately for odd and even values

% of rows and columns.

if mod(cols,2)

xrange = [-(cols-1)/2:(cols-1)/2]/(cols-1);

else

xrange = [-cols/2:(cols/2-1)]/cols;

end

if mod(rows,2)

yrange = [-(rows-1)/2:(rows-1)/2]/(rows-1);

else

yrange = [-rows/2:(rows/2-1)]/rows;

end

[x,y] = meshgrid(xrange, yrange);

radius = sqrt(x.^2 + y.^2); % Matrix values contain \*normalised\* radius from centre.

theta = atan2(-y,x); % Matrix values contain polar angle.

% (note -ve y is used to give +ve

% anti-clockwise angles)

radius = ifftshift(radius); % Quadrant shift radius and theta so that filters

theta = ifftshift(theta); % are constructed with 0 frequency at the corners.

radius(1,1) = 1; % Get rid of the 0 radius value at the 0

% frequency point (now at top-left corner)

% so that taking the log of the radius will

% not cause trouble.

sintheta = sin(theta);

costheta = cos(theta);

clear x; clear y; clear theta; % save a little memory

% Filters are constructed in terms of two components.

% 1) The radial component, which controls the frequency band that the filter

% responds to

% 2) The angular component, which controls the orientation that the filter

% responds to.

% The two components are multiplied together to construct the overall filter.

% Construct the radial filter components...

% First construct a low-pass filter that is as large as possible, yet falls

% away to zero at the boundaries. All log Gabor filters are multiplied by

% this to ensure no extra frequencies at the 'corners' of the FFT are

% incorporated as this seems to upset the normalisation process when

% calculating phase congrunecy.

lp = lowpassfilter([rows,cols],.45,15); % Radius .45, 'sharpness' 15

logGabor = cell(1,nscale);

for s = 1:nscale

wavelength = minWaveLength\*mult^(s-1);

fo = 1.0/wavelength; % Centre frequency of filter.

logGabor{s} = exp((-(log(radius/fo)).^2) / (2 \* log(sigmaOnf)^2));

logGabor{s} = logGabor{s}.\*lp; % Apply low-pass filter

logGabor{s}(1,1) = 0; % Set the value at the 0 frequency point of the filter

% back to zero (undo the radius fudge).

end

%% The main loop...

for o = 1:norient % For each orientation...

% Construct the angular filter spread function

angl = (o-1)\*pi/norient; % Filter angle.

% For each point in the filter matrix calculate the angular distance from

% the specified filter orientation. To overcome the angular wrap-around

% problem sine difference and cosine difference values are first computed

% and then the atan2 function is used to determine angular distance.

ds = sintheta \* cos(angl) - costheta \* sin(angl); % Difference in sine.

dc = costheta \* cos(angl) + sintheta \* sin(angl); % Difference in cosine.

dtheta = abs(atan2(ds,dc)); % Absolute angular distance.

% Scale theta so that cosine spread function has the right wavelength and clamp to pi

dtheta = min(dtheta\*norient/2,pi);

% The spread function is cos(dtheta) between -pi and pi. We add 1,

% and then divide by 2 so that the value ranges 0-1

spread = (cos(dtheta)+1)/2;

sumE\_ThisOrient = zero; % Initialize accumulator matrices.

sumO\_ThisOrient = zero;

sumAn\_ThisOrient = zero;

Energy = zero;

for s = 1:nscale, % For each scale...

filter = logGabor{s} .\* spread; % Multiply radial and angular

% components to get the filter.

% Convolve image with even and odd filters returning the result in EO

EO{s,o} = ifft2(imagefft .\* filter);

An = abs(EO{s,o}); % Amplitude of even & odd filter response.

sumAn\_ThisOrient = sumAn\_ThisOrient + An; % Sum of amplitude responses.

sumE\_ThisOrient = sumE\_ThisOrient + real(EO{s,o}); % Sum of even filter convolution results.

sumO\_ThisOrient = sumO\_ThisOrient + imag(EO{s,o}); % Sum of odd filter convolution results.

% At the smallest scale estimate noise characteristics from the

% distribution of the filter amplitude responses stored in sumAn.

% tau is the Rayleigh parameter that is used to describe the

% distribution.

if s == 1

if noiseMethod == -1 % Use median to estimate noise statistics

tau = median(sumAn\_ThisOrient(:))/sqrt(log(4));

elseif noiseMethod == -2 % Use mode to estimate noise statistics

tau = rayleighmode(sumAn\_ThisOrient(:));

end

maxAn = An;

else

% Record maximum amplitude of components across scales. This is needed

% to determine the frequency spread weighting.

maxAn = max(maxAn,An);

end

end % ... and process the next scale

% Accumulate total 3D energy vector data, this will be used to

% determine overall feature orientation and feature phase/type

EnergyV(:,:,1) = EnergyV(:,:,1) + sumE\_ThisOrient;

EnergyV(:,:,2) = EnergyV(:,:,2) + cos(angl)\*sumO\_ThisOrient;

EnergyV(:,:,3) = EnergyV(:,:,3) + sin(angl)\*sumO\_ThisOrient;

% Get weighted mean filter response vector, this gives the weighted mean

% phase angle.

XEnergy = sqrt(sumE\_ThisOrient.^2 + sumO\_ThisOrient.^2) + epsilon;

MeanE = sumE\_ThisOrient ./ XEnergy;

MeanO = sumO\_ThisOrient ./ XEnergy;

% Now calculate An(cos(phase\_deviation) - | sin(phase\_deviation)) | by

% using dot and cross products between the weighted mean filter response

% vector and the individual filter response vectors at each scale. This

% quantity is phase congruency multiplied by An, which we call energy.

for s = 1:nscale,

E = real(EO{s,o}); O = imag(EO{s,o}); % Extract even and odd

% convolution results.

Energy = Energy + E.\*MeanE + O.\*MeanO - abs(E.\*MeanO - O.\*MeanE);

end

%% Automatically determine noise threshold

%

% Assuming the noise is Gaussian the response of the filters to noise will

% form Rayleigh distribution. We use the filter responses at the smallest

% scale as a guide to the underlying noise level because the smallest scale

% filters spend most of their time responding to noise, and only

% occasionally responding to features. Either the median, or the mode, of

% the distribution of filter responses can be used as a robust statistic to

% estimate the distribution mean and standard deviation as these are related

% to the median or mode by fixed constants. The response of the larger

% scale filters to noise can then be estimated from the smallest scale

% filter response according to their relative bandwidths.

%

% This code assumes that the expected reponse to noise on the phase congruency

% calculation is simply the sum of the expected noise responses of each of

% the filters. This is a simplistic overestimate, however these two

% quantities should be related by some constant that will depend on the

% filter bank being used. Appropriate tuning of the parameter 'k' will

% allow you to produce the desired output.

if noiseMethod >= 0 % We are using a fixed noise threshold

T = noiseMethod; % use supplied noiseMethod value as the threshold

else

% Estimate the effect of noise on the sum of the filter responses as

% the sum of estimated individual responses (this is a simplistic

% overestimate). As the estimated noise response at succesive scales

% is scaled inversely proportional to bandwidth we have a simple

% geometric sum.

totalTau = tau \* (1 - (1/mult)^nscale)/(1-(1/mult));

% Calculate mean and std dev from tau using fixed relationship

% between these parameters and tau. See

% http://mathworld.wolfram.com/RayleighDistribution.html

EstNoiseEnergyMean = totalTau\*sqrt(pi/2); % Expected mean and std

EstNoiseEnergySigma = totalTau\*sqrt((4-pi)/2); % values of noise energy

T = EstNoiseEnergyMean + k\*EstNoiseEnergySigma; % Noise threshold

end

% Apply noise threshold, this is effectively wavelet denoising via

% soft thresholding.

Energy = max(Energy - T, 0);

% Form weighting that penalizes frequency distributions that are

% particularly narrow. Calculate fractional 'width' of the frequencies

% present by taking the sum of the filter response amplitudes and dividing

% by the maximum amplitude at each point on the image. If

% there is only one non-zero component width takes on a value of 0, if

% all components are equal width is 1.

width = (sumAn\_ThisOrient./(maxAn + epsilon) - 1) / (nscale-1);

% Now calculate the sigmoidal weighting function for this orientation.

weight = 1.0 ./ (1 + exp( (cutOff - width)\*g));

% Apply weighting to energy and then calculate phase congruency

PC{o} = weight.\*Energy./sumAn\_ThisOrient; % Phase congruency for this orientatio

pcSum = pcSum+PC{o};

% Build up covariance data for every point

covx = PC{o}\*cos(angl);

covy = PC{o}\*sin(angl);

covx2 = covx2 + covx.^2;

covy2 = covy2 + covy.^2;

covxy = covxy + covx.\*covy;

end % For each orientation

%% Edge and Corner calculations

% The following is optimised code to calculate principal vector

% of the phase congruency covariance data and to calculate

% the minimumum and maximum moments - these correspond to

% the singular values.

% First normalise covariance values by the number of orientations/2

covx2 = covx2/(norient/2);

covy2 = covy2/(norient/2);

covxy = 4\*covxy/norient; % This gives us 2\*covxy/(norient/2)

denom = sqrt(covxy.^2 + (covx2-covy2).^2)+epsilon;

M = (covy2+covx2 + denom)/2; % Maximum moment

m = (covy2+covx2 - denom)/2; % ... and minimum moment

% Orientation and feature phase/type computation

or = atan2(EnergyV(:,:,3), EnergyV(:,:,2));

or(or<0) = or(or<0)+pi; % Wrap angles -pi..0 to 0..pi

or = round(or\*180/pi); % Orientation in degrees between 0 and 180

OddV = sqrt(EnergyV(:,:,2).^2 + EnergyV(:,:,3).^2);

featType = atan2(EnergyV(:,:,1), OddV); % Feature phase pi/2 <-> white line,

% 0 <-> step, -pi/2 <-> black line

tY = conv2(pcSum.\*pcSum,ones(8),'valid');

Y = tY(1:8:end,1:8:end);

%%------------------------------------------------------------------

% CHECKARGS

%

% Function to process the arguments that have been supplied, assign

% default values as needed and perform basic checks.

function [im, nscale, norient, minWaveLength, mult, sigmaOnf, ...

k, cutOff, g, noiseMethod] = checkargs(arg)

nargs = length(arg);

if nargs < 1

error('No image supplied as an argument');

end

% Set up default values for all arguments and then overwrite them

% with with any new values that may be supplied

im = [];

nscale = 4; % Number of wavelet scales.

norient = 8; % Number of filter orientations.

minWaveLength = 3; % Wavelength of smallest scale filter.

mult = 2.1; % Scaling factor between successive filters.

sigmaOnf = 0.55; % Ratio of the standard deviation of the

% Gaussian describing the log Gabor filter's

% transfer function in the frequency domain

% to the filter center frequency.

k = 2.0; % No of standard deviations of the noise

% energy beyond the mean at which we set the

% noise threshold point.

cutOff = 0.5; % The fractional measure of frequency spread

% below which phase congruency values get penalized.

g = 10; % Controls the sharpness of the transition in

% the sigmoid function used to weight phase

% congruency for frequency spread.

noiseMethod = -1; % Choice of noise compensation method.

% Allowed argument reading states

allnumeric = 1; % Numeric argument values in predefined order

keywordvalue = 2; % Arguments in the form of string keyword

% followed by numeric value

readstate = allnumeric; % Start in the allnumeric state

if readstate == allnumeric

for n = 1:nargs

if isa(arg{n},'char')

readstate = keywordvalue;

break;

else

if n == 1, im = arg{n};

elseif n == 2, nscale = arg{n};

elseif n == 3, norient = arg{n};

elseif n == 4, minWaveLength = arg{n};

elseif n == 5, mult = arg{n};

elseif n == 6, sigmaOnf = arg{n};

elseif n == 7, k = arg{n};

elseif n == 8, cutOff = arg{n};

elseif n == 9, g = arg{n};

elseif n == 10,noiseMethod = arg{n};

end

end

end

end

% Code to handle parameter name - value pairs

if readstate == keywordvalue

while n < nargs

if ~isa(arg{n},'char') || ~isa(arg{n+1}, 'double')

error('There should be a parameter name - value pair');

end

if strncmpi(arg{n},'im' ,2), im = arg{n+1};

elseif strncmpi(arg{n},'nscale' ,2), nscale = arg{n+1};

elseif strncmpi(arg{n},'norient' ,4), norient = arg{n+1};

elseif strncmpi(arg{n},'minWaveLength',2), minWaveLength = arg{n+1};

elseif strncmpi(arg{n},'mult' ,2), mult = arg{n+1};

elseif strncmpi(arg{n},'sigmaOnf',2), sigmaOnf = arg{n+1};

elseif strncmpi(arg{n},'k' ,1), k = arg{n+1};

elseif strncmpi(arg{n},'cutOff' ,2), cutOff = arg{n+1};

elseif strncmpi(arg{n},'g' ,1), g = arg{n+1};

elseif strncmpi(arg{n},'noiseMethod' ,4), noiseMethod = arg{n+1};

else error('Unrecognised parameter name');

end

n = n+2;

if n == nargs

error('Unmatched parameter name - value pair');

end

end

end

if isempty(im)

error('No image argument supplied');

end

if ndims(im) == 3

warning('Colour image supplied: converting image to greyscale...')

im = double(rgb2gray(im));

end

if ~isa(im, 'double')

im = double(im);

end

if nscale < 1

error('nscale must be an integer >= 1');

end

if norient < 1

error('norient must be an integer >= 1');

end

if minWaveLength < 2

error('It makes little sense to have a wavelength < 2');

end

if cutOff < 0 || cutOff > 1

error('Cut off value must be between 0 and 1');

end

%%-------------------------------------------------------------------------

% RAYLEIGHMODE

%

% Computes mode of a vector/matrix of data that is assumed to come from a

% Rayleigh distribution.

%

% Usage: rmode = rayleighmode(data, nbins)

%

% Arguments: data - data assumed to come from a Rayleigh distribution

% nbins - Optional number of bins to use when forming histogram

% of the data to determine the mode.

%

% Mode is computed by forming a histogram of the data over 50 bins and then

% finding the maximum value in the histogram. Mean and standard deviation

% can then be calculated from the mode as they are related by fixed

% constants.

%

% mean = mode \* sqrt(pi/2)

% std dev = mode \* sqrt((4-pi)/2)

%

% See

% http://mathworld.wolfram.com/RayleighDistribution.html

% http://en.wikipedia.org/wiki/Rayleigh\_distribution

%

function rmode = rayleighmode(data, nbins)

if nargin == 1

nbins = 50; % Default number of histogram bins to use

end

mx = max(data(:));

edges = 0:mx/nbins:mx;

n = histc(data(:),edges);

[dum,ind] = max(n); % Find maximum and index of maximum in histogram

rmode = (edges(ind)+edges(ind+1))/2;

% LOWPASSFILTER - Constructs a low-pass butterworth filter.

%

% usage: f = lowpassfilter(sze, cutoff, n)

%

% where: sze is a two element vector specifying the size of filter

% to construct [rows cols].

% cutoff is the cutoff frequency of the filter 0 - 0.5

% n is the order of the filter, the higher n is the sharper

% the transition is. (n must be an integer >= 1).

% Note that n is doubled so that it is always an even integer.

%

% 1

% f = --------------------

% 2n

% 1.0 + (w/cutoff)

%

% The frequency origin of the returned filter is at the corners.

%

% See also: HIGHPASSFILTER, HIGHBOOSTFILTER, BANDPASSFILTER

%

% Copyright (c) 1999 Peter Kovesi

% School of Computer Science & Software Engineering

% The University of Western Australia

% http://www.csse.uwa.edu.au/

%

% Permission is hereby granted, free of charge, to any person obtaining a copy

% of this software and associated documentation files (the "Software"), to deal

% in the Software without restriction, subject to the following conditions:

%

% The above copyright notice and this permission notice shall be included in

% all copies or substantial portions of the Software.

%

% The Software is provided "as is", without warranty of any kind.

% October 1999

% August 2005 - Fixed up frequency ranges for odd and even sized filters

% (previous code was a bit approximate)

function f = lowpassfilter(sze, cutoff, n)

if cutoff < 0 | cutoff > 0.5

error('cutoff frequency must be between 0 and 0.5');

end

if rem(n,1) ~= 0 | n < 1

error('n must be an integer >= 1');

end

if length(sze) == 1

rows = sze; cols = sze;

else

rows = sze(1); cols = sze(2);

end

% Set up X and Y matrices with ranges normalised to +/- 0.5

% The following code adjusts things appropriately for odd and even values

% of rows and columns.

if mod(cols,2)

xrange = [-(cols-1)/2:(cols-1)/2]/(cols-1);

else

xrange = [-cols/2:(cols/2-1)]/cols;

end

if mod(rows,2)

yrange = [-(rows-1)/2:(rows-1)/2]/(rows-1);

else

yrange = [-rows/2:(rows/2-1)]/rows;

end

[x,y] = meshgrid(xrange, yrange);

radius = sqrt(x.^2 + y.^2); % A matrix with every pixel = radius relative to centre.

f = ifftshift( 1.0 ./ (1.0 + (radius ./ cutoff).^(2\*n)) ); % The filter

1. **Train\_fcm function:**

for yi = 1: 40

yi

%% To read a image from file...

I = imread(['Data\_sets\',num2str(yi),'.jpg']);

%% Grayscale conversion..

I = rgb2gray(I);

%% Gaussian filtering..

F = fspecial('gaussian');

I = imfilter(I,F);

figure(3); imshow(I); title('Gaussian Filtered Image');

Gray = I;

%% FCM Segmentation..

fim=mat2gray(I);

level=graythresh(fim);

bwfim=im2bw(fim,level);

[bwfim0,level0]=fcmthresh(fim,0);

[seg\_img,level1]=fcmthresh(fim,1);

%% Haar Feature Extraction..

[cA,cH,cV,cD] = dwt2(seg\_img,'haar');

cA = mean2(cA);

cH = mean2(cH);

cV = mean2(cV);

cD = mean2(cD);

Mean = mean2(seg\_img);

Standard\_Deviation = std2(seg\_img);

Entropy = entropy(seg\_img);

Variance = mean2(var(double(seg\_img)));

a = sum(double(seg\_img(:)));

Smoothness = 1-(1/(1+a));

Kurtosis = kurtosis(double(seg\_img(:)));

Skewness = skewness(double(seg\_img(:)));

% Inverse Difference Movement

m = size(seg\_img,1);

n = size(seg\_img,2);

in\_diff = 0;

for i = 1:m

for j = 1:n

temp = seg\_img(i,j)./(1+(i-j).^2);

in\_diff = in\_diff+temp;

end

end

IDM = double(in\_diff);

featext\_fcm = [Mean, Standard\_Deviation, Entropy,Variance, Smoothness, Kurtosis, Skewness, IDM, cA,cH,cV,cD];

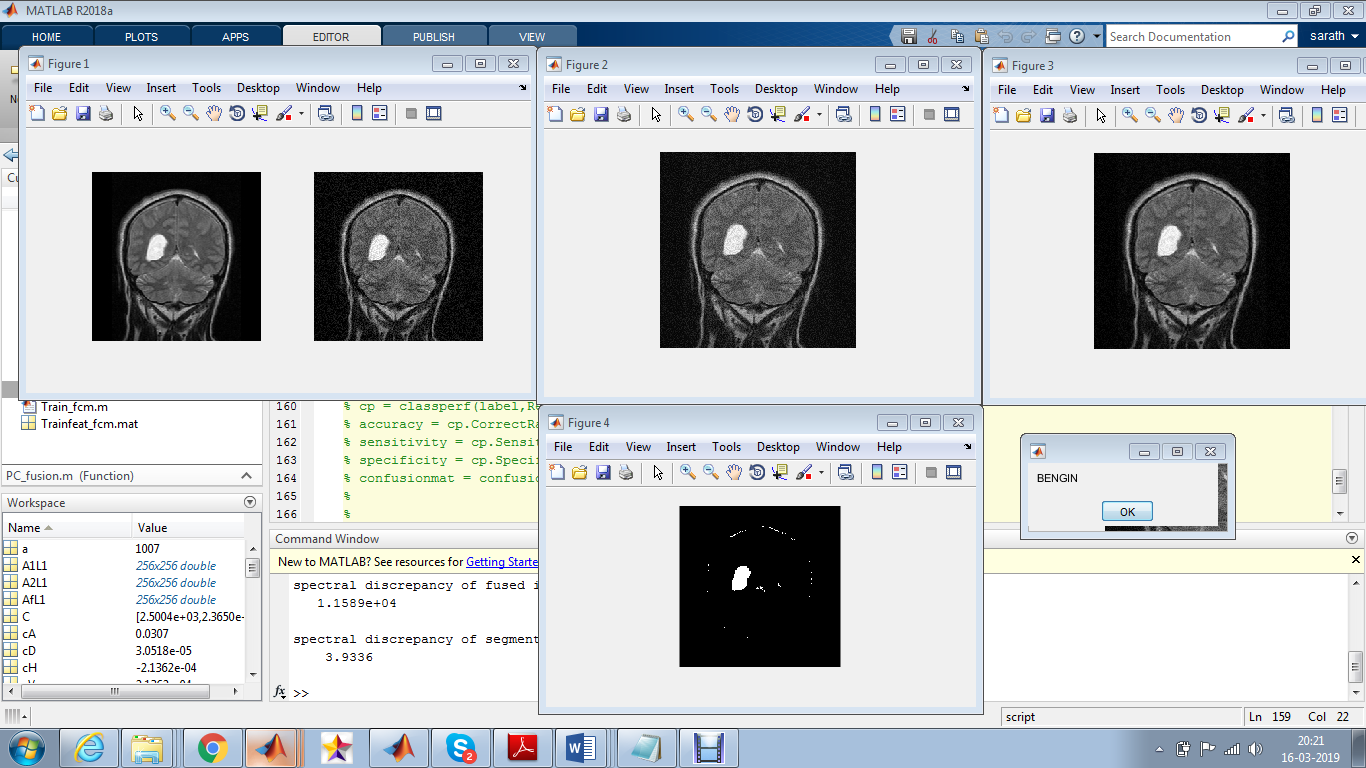
Trainfeat\_fcm(yi,:) = featext\_fcm;

end

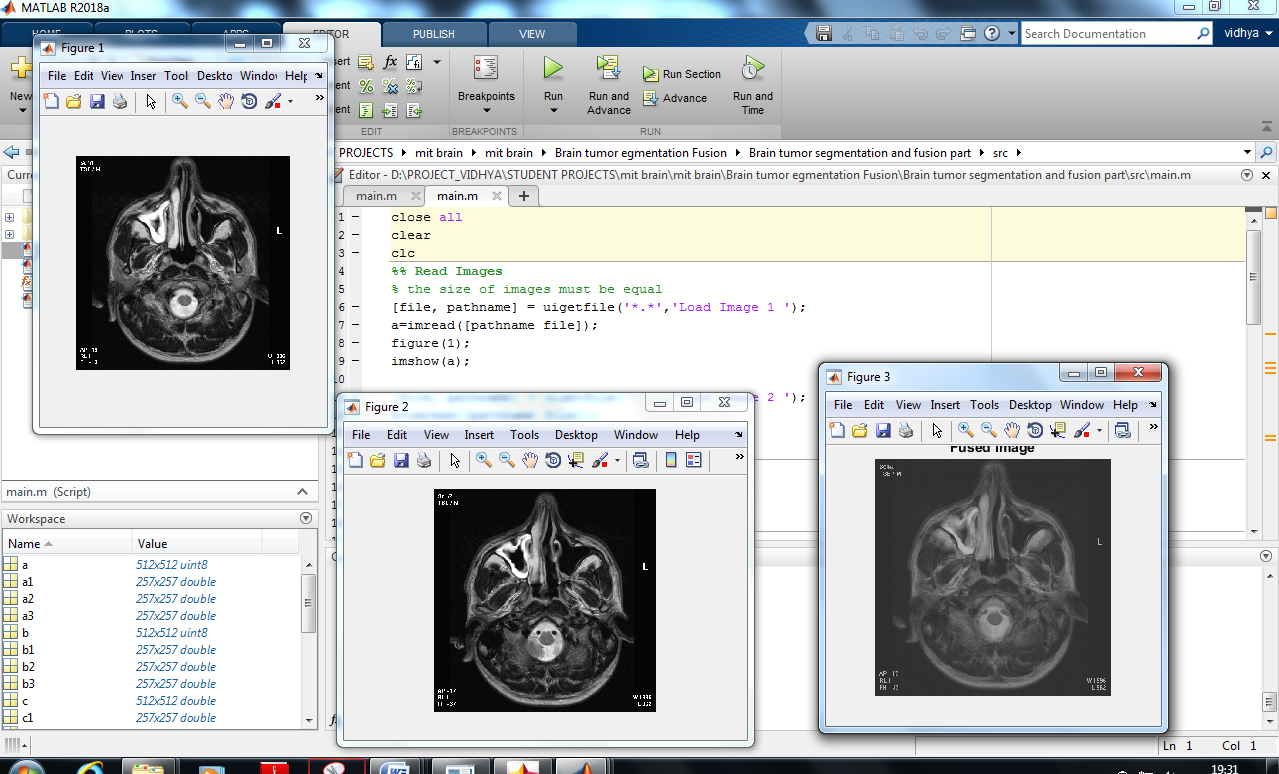
save Trainfeat\_fcm Trainfeat\_fcm

**Screenshots:**

**Sceenshot 1:**

****

**Screenshot 2:**

****

**CONCLUSION**

Medical image fusion combines different modality of medical images to produce a high quality fused image with spatial and spectral information. The fused image with more information improved the performance of image analysis algorithms used in different medical diagnosis applications.SVM is used in this paper for brain image fusion and K-Clustering features are extracted from the brain image. The brain tumor region is segmented using the extracted features and adaptive SVM classifier helps to identify whether the tumor is benign or malignant. Thus it helps the physician and radiologist for brain tumor diagnosis for human surgery.

**FUTURE ENCHANCEMENT**

In future work, it would be interesting to include additional feature information. Besides the energy, correlation, contrast and homogeneity add more information to the feature extraction in order to make the system more sensitive; information from the textures or location. It will be interesting to continue developing more adaptive models for other types of brain tumors following the same line of work presented here. Another future line would be the detection of small malignant brain tumors. It should be clear that many factors influence the appearance of tumors on images, and although there are some common features of malignancies, there is also a great deal of variation that depends on the tissue and the tumor type. Characteristic features are more likely to be found in large tumors. Small tumors may not have many of the features of malignancy and may even manifest themselves only by secondary effects such as architectural distortion.

**REFERENCES:**

[1] T. Zaveri, and M. Zaveri, “A Novel Region Based Multimodality Image Fusion Method”, Journal of Pattern Recognition Research, vol. 2, pp.

140–153, 2011.

[2] V. Tsagaris, V. Anastassopoulos, and G. Lampropoulos, “Fusion of hyperspectral data using segmented PCT for enhanced color representation”, IEEE Trans. Geosci. Remote Sens., vol. 43, no. 10, pp. 2365–2375, 2005.

[3] G. Bhatnagar, Q. M. J. Wu, and Z. Liu, “Directive Contrast Based Multimodal Medical Image Fusion in NSCT Domain. IEEE Transactions on Multimedia, vol. 15, no. 5, pp. 1014–1024, 2013. DOI:

10.1109/TMM.2013.2244870.

[4] A. Rana, and S. Arora, “Comparative Analysis of Medical Image Fusion. International Journal of Computer Applications,” vol. 73, no. 9, pp.10–13, 2013. DOI: 10.5120/12768-9371.

[5] Pinki Jain, and Anu Aggarwal, “Text Fusion in Medical Images using Fuzzy Logic based Matrix Scanning Algorithm,” International Journal of Scientific and Research Publications, vol. 2, no. 6, pp. 1-6, 2012.

[6] Devid Hall, James LLians, Hand book of multisensor data fusion, CRC Press LLC, 2001.

[7] Goshtasby, Image fusion: Advances in the state of the art Journal of Information Fusion, Elsevier, Vol. 8, pp 114-118, 2007.

[8] A general framework for multiresolution image fusion: from pixels to regions, Journal of Information Fusion, Vol. 4 (4), pp 259-280, 2003.

[9] P. J. Burt, E. H. Adelson, Mergin images through pattern decomposion, Proc. SPIE 575, pp 173-181, 1985.

[10] A. Toet, Hierarchical Image Fusion, Machine Vision and Applications, Vol. 3, pp 1-11, 1990.

[11] Anna Wang, Jaijining Sun and Yueyang Guan, The application to wavelet transform to multimodality medical image fusion, IEEE International Conference on Networking, Sensing and Control, pp. 270-274, 2006.

[12] Miao Qiguang and Wang Baoshul, A novel image fusion method using contourlet transform, International Conference on Communications, Circuits and Systems Proceedings, Vol. 1, pp 548-552, 2006.

[13] Zhong Zhang and Rick Blum, A categorization of multiscale decomposition based image fusion schemes with a performance study for digital camera application, Proceedings of IEEE, Vol. 87 (8), pp 1315- 1326, Aug 1999.

[14] Gemma Piella, A region based multiresolution image fusion algorithm, Proceedings of the Fifth International Conference on Information Fusion, Vol. 2, pp 1557- 1564, 2002

[15] J. Lewis, R. J. O’Callaghan, S. G. Nikolov, D.R. Bull, C. N. Canagarajah, Region based image fusion using complex wavelets, Proceedings of the 7th International on Information fusion, Stockholm, Sweden, pp 555-562, July, 2004

[16] Shutao Li and Bin Yang, Multifocus image fusion using region segmentation and spatial frequency, Image and Vision Computing. Elsevier, Vol. 26, pp. 971-979, 2008

[17] J. Shi and J. Malik, Normalized cuts and image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22 (8), 888-905, 2000.

[18] Tanish Zaveri, Mukesh Zaveri, A Novel Region Based Multifocus Image Fusion Method, IEEE International Conference on digital image processing, Bangkok, Thailand, pp. 50-55, March 2009

[19] Tanish Zaveri, Mukesh Zaveri, Region Based Image Fusion for Detection of Ewing Sarcoma, International Conference on advances in pattern reorganization, ISI, Kolkata, pp 240-243, Feb. 2009

[20] Kayan Najarian, Robert Splinter, Biomedical Signal and Image processing, CRC Press, pp. 60-61, 2006

[21] Rafael Gonzalez, Richard Woods, Digital Image Processing, Pearson Education, 2nd ed., 2006

[22] C.S. Xydeas and V. Petrovic, Objective image fusion performance measure, Electronics Letters, Vol. 36 (4), pp 308-309, February 2000

[23] Yin Chen, Rick S. Blum, A automated quality assessment algorithm for image fusion, Image and Vision Computing, Elsevier, 2008

[24] Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Transactions on Image Processing Vol. 13, 600-612, 2004

[25] Zheng Liu and Robert Laganiere, On the use of phase congruency to evaluate image similarity, IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, Vol. 2, pp 937-940, 2006

[26] S.G. Mallat, A theory for multiresoultuion signal decomposition: the wavelet representation, IEEE Trans. On Pattern Analysis and Machine Intelligence, Vol. 2(7), pp. 674-693, July 1989

[27] Zhou Wang, Alan Bovik, A universal image quality index, IEEE signal processing letters, Vol. 9, pp.

81-81, Mar. 2002

[28] Timothee Cour, Florence Benezit, Jianbo Shi, Multiscale Normalized Cuts Segmentation Toolbox for MATLAB, [online] Available : http://www.seas.upenn.edu/ timothee.

[29] Rui Xu, Donald Wunsch, , Survey of Clustering Algorithms IEEE Transactions On Neural Networks, Vol. 16 (3), pp. 645-678, May 2005.

[30] Rao R.M., Bopardikar A.S., Wavelet Transforms: Introduction to theory and applications, Pearson Education India, 2000.