EFFECTIVE BRAIN TUMOUR DETECTION USING MODERN ML APPROACHES

A Project work (A80088)

Submitted

in partial fulfilment of the requirements for

the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Mr. Aniket Mahindrakar
H. T. No: 15261A0507

Mr. Karamcheti Sri Krishna Manoj
H. T. No: 15261A0529

Under the Guidance of

Dr. A Nagesh
(Professor)

Mr. R Srinivas
(Sr. Asst. Professor)

Department of Computer Science and Engineering

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(Affiliated to JNTU, Hyderabad; Accredited by NBA, AICTE-New Delhi)

Kokapet(vill), Rajendra Nagar (Mandal), Ranga Reddy(Dist.),

Chaitanya Bharathi P.O., Hyderabad - 500 075, India

APRIL 2019

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(Established in 1997 by Chaitanya Bharathi Educational Society)

(Affiliated to JNTU, Hyderabad; Accredited by NBA, AICTE-New Delhi)

Kokapet(village & gram panchayat), Rajendra Nagar (Mandal), Ranga Reddy(Dist.),

Chaitanya Bharathi P.O., Hyderabad-500 075.

CERTIFICATE

This is to certify that the project titled ***Effective Brain Tumour Detection using Modern ML Approaches***, being submitted by **Mr. Aniket Mahindrakar** and **Mr. Karamcheti Sri Krishna Manoj** bearing **Roll No: 15261A0507** and **15261A0529** respectively in partial fulfilment for the award of **B. Tech** in Computer Science and Engineering to Mahatma Gandhi Institute of Technology, Gandipet, Hyderabad is a record of bonafide work carried out by them under our guidance and supervision.

The results embodied in this project have not been submitted to any other University or Institute for the award of any degree or diploma.

Project Guide	**Project Guide**	**Head of the Department**
Dr. A Nagesh	**Mr. R Srinivas**	**Dr. C.R.K. Reddy**
Professor	Sr. Asst. Prof	Professor
Dept. of CSE	Dept. of CSE	Dept. of CSE
MGIT, Hyderabad	MGIT, Hyderabad	MGIT, Hyderabad

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the work presented in this project, titled ***Effective Brain Tumour Detection Using Modern ML Approaches***, which is being submitted by us in partial fulfilment for the award of B. Tech in the Department of Computer Science and Engineering to Mahatma Gandhi Institute of Technology, Gandipet, Hyderabad (T.S) - 500075, is a result of investigations carried out by us under the guidance of **Dr. A Nagesh**, Professor of Computer Science and Engineering, MGIT, Gandipet, Hyderabad and **Mr. R Srinivas,** Sr. Asst. Prof of Computer Science and Engineering, MGIT, Gandipet, Hyderabad.

The work is original and has not been submitted for any Degree/Diploma of this or any other university.

Place: Hyderabad
Date: 20 – 04 – 2019

Name of the Candidate: Aniket Mahindrakar
Roll No: 15261A0507

Name of the Candidate: K. Sri Krishna Manoj
Roll No: 15261A0529

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to our advisors, **Dr. A Nagesh** and **Mr. R Srinivas,** whose knowledge and guidance has motivated us to achieve goals we never thought possible. They have consistently been a source of motivation, encouragement and inspiration. The time we have spent working under their supervision has truly been a pleasure.

We would like to thank **Ms. P Poornima** and **Mr. A Ratna Raju** for their valuable inputs in improving several aspects of this project.

We profoundly thank our Head of the Department of Computer Science and Engineering **Dr. C.R.K. Reddy** for his constant support and inspiration. We thank our guides and all faculty members of Department of CSE for their help during our course. Thanks to Teaching and Non-Teaching staff of the Departments of CSE. We thank our Principal **Dr. K. Jaya Sankar** for encouraging us to pursue this project.

Finally, special thanks to our family members for their support and encouragement throughout our course. Thanks to all our friends and well wishers for their constant support.

Aniket Mahindrakar
H. T. No: 15261A0507
Karamcheti Sri Krishna Manoj
H.T. No: 15261A0529

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Brain tumour analysis from MR Images is one of the major issues that doctors face on a regular basis. The brain tumour analysis involves grading of brain tumours. Manual detection of brain tumours is time-consuming and error prone. Thus, automated techniques for brain tumour detection are necessary. This project aims to evaluate various classifiers to detect brain tumour from MR Images. Based on the results obtained, one can decide which model gives the best results. The dataset consists of 3064 T1-weighted Brain tumour MR images taken from 233 patients with three kinds of brain tumour: Meningioma, Glioma and Pituitary Tumour. The classifiers employed are: Decision Tree, Support Vector Machine (SVM), Convolutional Neural Networks (CNN) and Decision Tree Ensembles – Bagging.

1. Introduction

Brain is a complex organ which is made up of many tissues. A tissue is a collection of similar type of cells. When these tissues grow abnormally and form a mass, they are called as tumours. Tumours are broadly classified into two types: benign and malignant. Benign tumours are those which do not transform into cancer. Malignant tumours are those tumours which may transform into cancer and may prove to be fatal.

The symptoms that accompany a brain tumour include: headaches, difficulty in walking and speaking, difficulty in coordination, seizures, nausea or vomiting and difficulty in speaking. Mental confusion may present itself as one of the symptoms. Behaviour and personality changes may also be observed in patients.

Brain tumours are primarily classified into three types:

i. Glioma: This tumour starts in glial cells of the brain or the spine.
ii. Meningioma: This tumour forms in the meninges, which is a membranous part surrounding the brain and the spine.
iii. Pituitary adenomas: This tumour occurs in the pituitary gland.

Diagnosing of brain tumour can be done by imaging techniques such as Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) scans. Apart from these techniques, histological methods such as biopsy are performed on tissue samples for confirmation.

The treatment options available for patients suffering from brain tumour include:

i. Surgery: Removing the tumour part in the brain with the help of various medical procedures.
ii. Radiotherapy: Usage of high energy waves such as beta or gamma rays to treat brain tumours.
iii. Chemotherapy: Usage of anti cancer drugs such as Temozolomide, Procarbazine, Lomustine etc. to destroy cancer cells.

Manual detection of brain tumours is time-consuming and error prone. Thus, automated techniques for brain tumour detection are necessary. Many automated techniques for detection of brain tumours have been proposed. These techniques use classifiers such as Support Vector Machines, Decision Trees, KNN classifiers and Convolutional Neural Networks (CNNs). Automated techniques are relatively fast and more accurate when compared to Manual segmentation.

1.1 Existing System

There are already many models that have been built for the purpose of classification of Brain Tumours. A flow chart given below illustrates the existing system design (Fig 1.1):

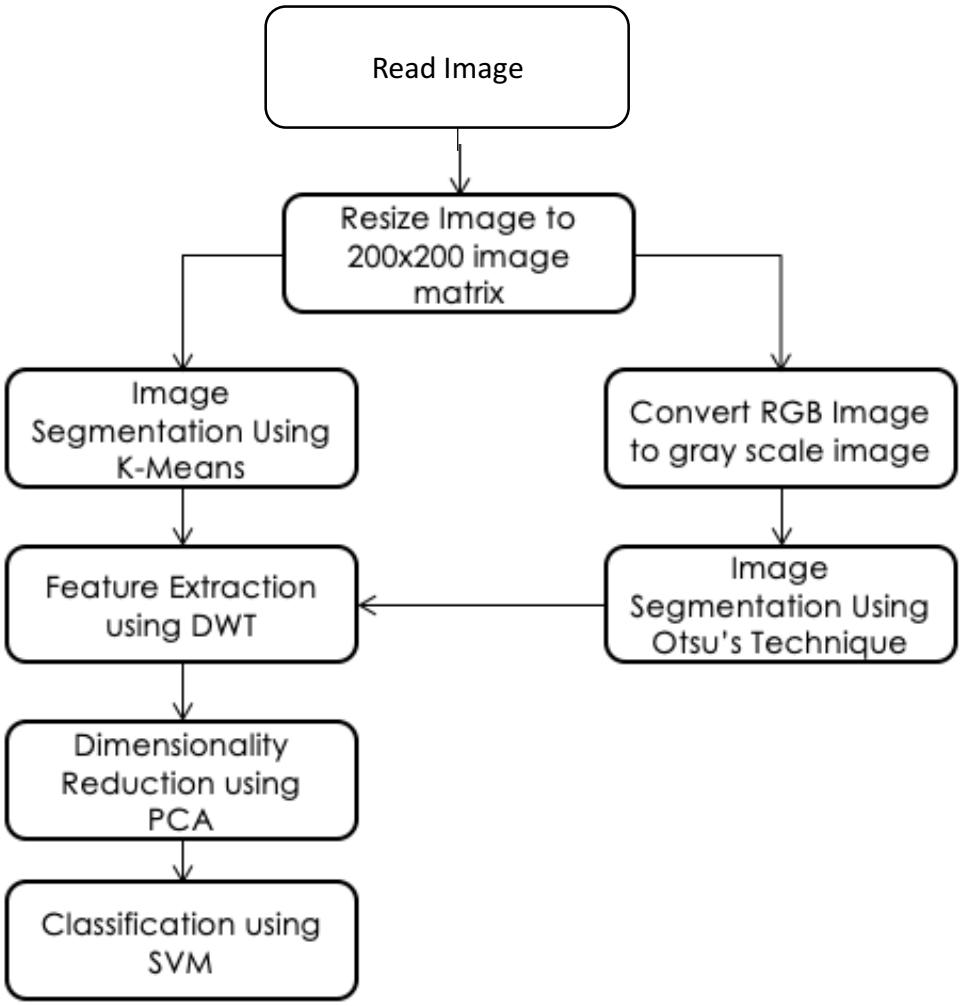

Fig 1.1: Existing System Design

Most of the times, image segmentation was done using Otsu's technique or K-means clustering technique. Feature extraction was done using Discrete Wavelet Transform (DWT) and the dimensionality reduction was done using Principal Component Analysis (PCA). It was observed that most of researchers tend to prefer SVM for classification of brain tumours.

1.2 Proposed System

In the proposed system, instead of restricting to Support Vector Machine (SVM), a comparison of performance of various models is provided so that it is left to the discretion of the user as to which model he/she must choose. The proposed system (Fig 1.2) would use the feature extraction using DWT and dimensionality reduction using PCA as in existing system.

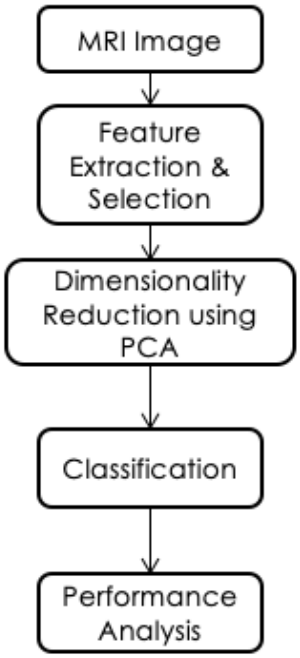

Fig 1.2: Proposed System Design

1.3 Requirements Specification

1.3.1 Software Requirements

- Operating System: Windows 7 or later, macOS X, Ubuntu 14 or later
- MATLAB r2017a or later

1.3.2 Hardware Requirements

- Processor: Intel core i3 or higher
- Hard Disk: 5 Gb or more
- RAM: 4 Gb or more
- NVIDIA Graphic card with Cuda

1.3.3 MATLAB Toolbox Requirements

- Image Processing Toolbox
- Deep learning toolbox
- MATLAB GPU Computing Support

1.3.4 Functional Requirements

- The project demonstrates development of various models for training and classification
- Preparing training and test data
- Data Augmentation
- Setting various parameters for models that are being developed
- Fitting the models

1.3.5 Non Functional Requirements

Usability

- The system should provide a comparison of accuracies given by various classifiers.
- The user should be able to test the working of the classifiers by providing his/her own images.

Reliability

- The system will be developed using various machine learning and deep learning methods. The accuracies given by each model on the validation set gives the reliability percentage of the model.

Performance

- A variation of ±3% in accuracies should be acceptable.

2. Literature Survey

In 2017, Bahadure et al. published a paper titled "Image Analysis for MRI Based Brain Tumour Detection and Feature Extraction using Biologically inspired BWT and SVM"[1] in which Support Vector Machine (SVM) was used for classification. Additionally, skull stripping algorithm based on threshold technique was employed. It improved signal to noise ratio. However, it resulted in high accuracy due to overfitting of the data.

Manu BN et al. proposed a method for brain tumour detection and classification[2]. Again, Support Vector Machine (SVM) was used. Discrete Wavelet Transform and Principal Component Analysis for feature extraction and dimensionality reduction respectively were used. Detailed comparison between linear, quadratic and polynomial kernel SVM accuracies was provided. However, it suffered from overfitting due to small training set.

Chandra Sekhar Ravuri proposed a method for automatic segmentation of brain tumour[3]. A 1-class SVM classifier was used. Anisotropic diffusion filter was used to remove noise. Fast bounding box algorithm was employed for image segmentation. This precision achieved high precision and dependability. However, the process was not fully automated.

In 2015, Cheng J et al. published a paper titled "Enhanced Performance of Brain Tumour Classification via Tumour Region Augmentation and Partition"[4]. In this publication, the authors made use of intensity normalization using intensity histogram, GLCM and Raw patch based Bag of Words (BoW) model. The authors augmented the tumour region and made use of large dataset. However, no noise removal was performed and features were extracted from corrupted images as well.

In 2018, Naskar et al. published a paper titled "Automated System for brain Tumour Detection and Segmentation"[5]. Threshold and watershed segmentation were used. K-means to find the density of tumour. Classification can be done using any image (ex. MRI, CT scans etc.). The tumour portion was detected but there were still holes in the interior of the tumour cell.

In 2018, Ari et al. proposed deep learning methods for classification and detection[6]. The authors used Used Extreme Learning Machines – Local Receptive Fields. This method again suffered from overfitting.

The above information has been summarised in Table 2.1.

S.No	Year	Authors	Title	Techniques	Advantages	Disadvantages
1.	2017	Nilesh Bhaskarrao Bahadure, Arun Kumar Ray, and Har Pal Thethi	Image Analysis for MRI Based Brain Tumour Detection and Feature Extraction using Biologically inspired BWT and SVM	Support Vector Machine (SVM) for classification. Skull stripping algorithm based on threshold technique.	Improved signal to noise ratio	High accuracies might have resulted from overfitting of the data.
2.	2017	Manu BN	Brain MRI Tumour Detection and Classification	Support Vector Machine (SVM). DWT and PCA for feature extraction and dimensionality reduction respectively.	Detailed comparison between linear, quadratic and polynomial kernel SVM accuracies.	Small training and test set which might have resulted in overfitting.
3.	2015	Chandra Sekhar Ravuri	Automatic Segmentation of Brain Tumour	1-class SVM classifier. Using Anisotropic diffusion filter to remove noise. Fast bounding box algorithm for image segmentation.	High precision and dependability	The process is not fully automated. Works only on few images. Not effective

4.	2015	Cheng J, Huang W, Cao S, Yang R, Yang W	Enhanced Performance of Brain Tumour Classification via Tumour Region Augmentation and Partition	Intensity normalization using Intensity Histogram, GLCM and Raw patch based BoW model.	Augmenting the tumour region. Large dataset used.	No noise removal performed and extracted features from corrupted images as well.
5.	2018	Madhumantee Naskar	Automated System for brain Tumour Detection and Segmentation	Threshold and watershed segmentation. K-means to find the density of tumour.	Classification can be done using any image	The tumour portion is detected but there are still holes in the interior of the tumour cell.
6.	2018	Ali Ari, Davut Hanbay	Deep learning based brain tumour classification and detection system	Used Extreme Learning Machines – Local Receptive Fields	Usage of deep learning methods	Small training and test set which might have resulted in overfitting.

Table 2.1: Literature Survey of various papers published on Brain Tumour Detection

3. System Design of Effective Brain Tumour Detection Using Modern ML Approaches

3.1 UML Diagrams

3.1.1 Use Case Diagram

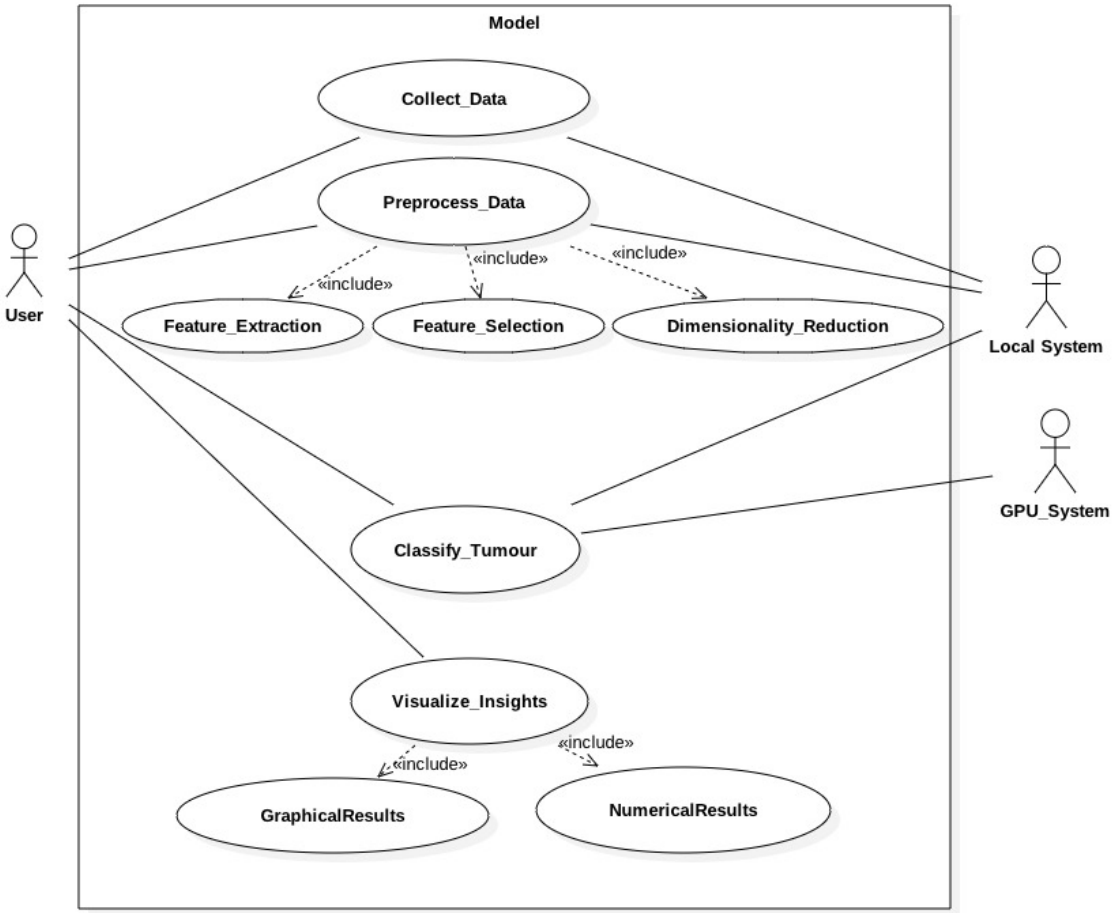

Fig 3.1: Use Case Diagram for Effective Brain Tumour Detection

A user who is using the system, can collect the data from the local system, pre-process it and send the tumour for classification. After classification, the user can visualize the data either graphically or numerically as shown in the use case diagram (Fig 3.1). The GPU comes into the picture when the classification is done using Convolutional Neural Network (CNN). Pre-processing involves feature extraction, feature selection and dimensionality reduction.

3.1.2 Activity diagram

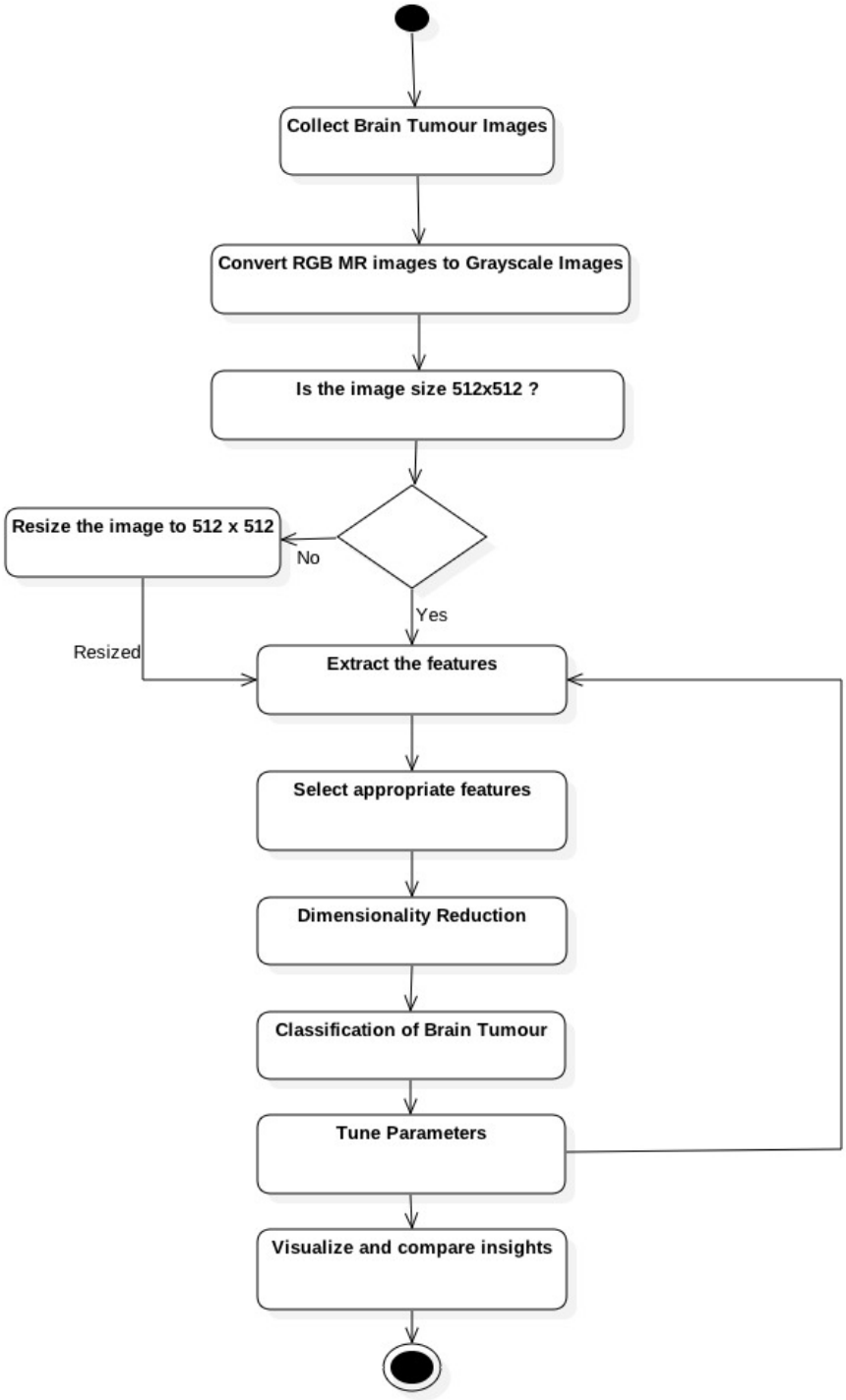

Fig 3.2: Activity Diagram for Brain Tumour Detection and Classification

The activity diagram (Fig 3.2) is similar to the flow chart of the proposed system. In addition to it, we tune parameters continuously to get the optimized results for each model.

3.1.3 Deployment Diagram

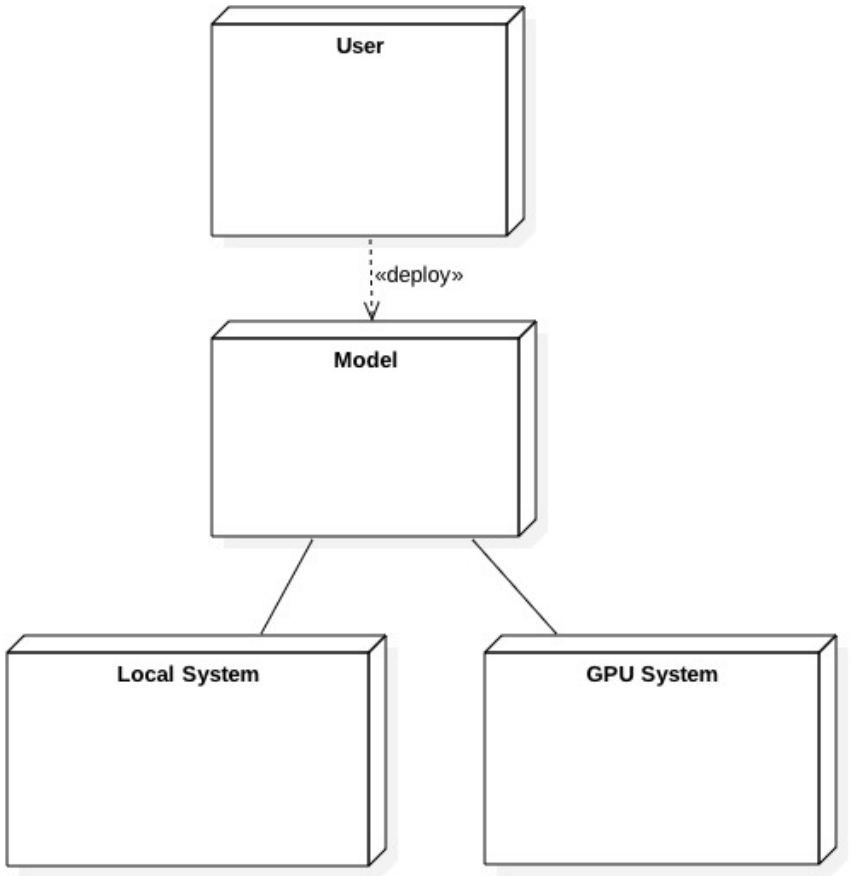

Fig 3.3: Deployment Diagram for Brain Tumour Detection and Classification

The user deploys the model (Fig 3.3), built using the training data. The model communicates with the local system for training and test data fetching. The model communicates with GPU system for additional computation power.

3.2 Modules

3.2.1 Data Collection Module

In this module, the data is loaded from the required folder using load() function. In the dataset that is used, there might be some corrupt images (those which are not of 512x512 in size). Such images have to be resized before we can use them further. The dataset that is used consists of .mat files. Files with a .mat extension contain MATLAB formatted data, and data can be loaded from or written to these files using the functions load and save, respectively.

3.2.2 Data Pre-processing Module

The Data Pre-processing module consists of 3 sub-modules:

1. Feature Extraction Module
2. Dimensionality Reduction Module
3. Feature Selection Module

Feature Extraction is done using Discrete Wavelet transform (DWT). The discrete wavelet transform (DWT) is a linear signal processing technique that, when applied to a data vector X, transforms it to a numerically different vector, X′, of wavelet coefficients. The two vectors are of the same length[7]. The dwt2() command performs a single-level two-dimensional wavelet decomposition.

Dimensionality reduction is the process of reducing the number of random variables or attributes under consideration[7]. Principal Components Analysis (PCA) searches for k n-dimensional orthogonal vectors that can best be used to represent the data, where $k \leq n$. The original data are thus projected onto a much smaller space, resulting in dimensionality reduction. pca(X) returns the principal component coefficients, also known as loadings, for the n-by-p data matrix X. Rows of X correspond to observations and columns correspond to variables. The coefficient matrix is p-by-p matrix. Principal Components Analysis helps to reduce correlated variables/features of dataset from huge number to lesser uncorrelated variables. It does this by constructing k n-dimensional orthogonal unit vectors that best describe the data. The data is then projected onto these orthogonal vectors, also known as principal components, without loosing much of the information. Features that are less significant and informative get discarded in the process and features which explain data more are retained.

In the Feature Selection Module, we form a list of features that are to be selected, to be used in the subsequent steps as data for training of models. The features selected are listed below:

i. Contrast
ii. Correlation
iii. Energy
iv. Homogeneity
v. Mean
vi. Standard Deviation
vii. Entropy

viii. Root Mean Square level (RMS)
ix. Variance
x. Smoothness
xi. Kurtosis
xii. Skewness
xiii. Inverse Difference Movement

3.2.3 Classification Module

In this module, the tumour images will be classified by different classifiers into one of the following three classes:

1. Glioma
2. Meningioma
3. Pituitary tumour

The classifiers chosen are:

i. Support Vector Machine (SVM)
ii. Decision Trees
iii. Ensembles (Bagging)
iv. Convolutional Neural Network (CNN)

Support Vector Machine (SVM) uses a mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane (i.e., a "decision boundary" separating the tuples of one class from another). With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using support vectors ("essential" training tuples) and margins (defined by the support vectors)[7]. Although the training time of even the fastest SVMs can be extremely slow, they are highly accurate, owing to their ability to model complex nonlinear decision boundaries. They are much less prone to overfitting than other methods. The support vectors found also provide a compact description of the learned model. SVMs can be used for numeric prediction as well as classification. They have been applied to a number of areas, including handwritten digit recognition, object recognition, and speaker identification, as well as benchmark time-series prediction tests[9]. For linearly separable data, the SVM works by searching for a maximal margin hyperplane (MMH) as shown in Fig 3.4.

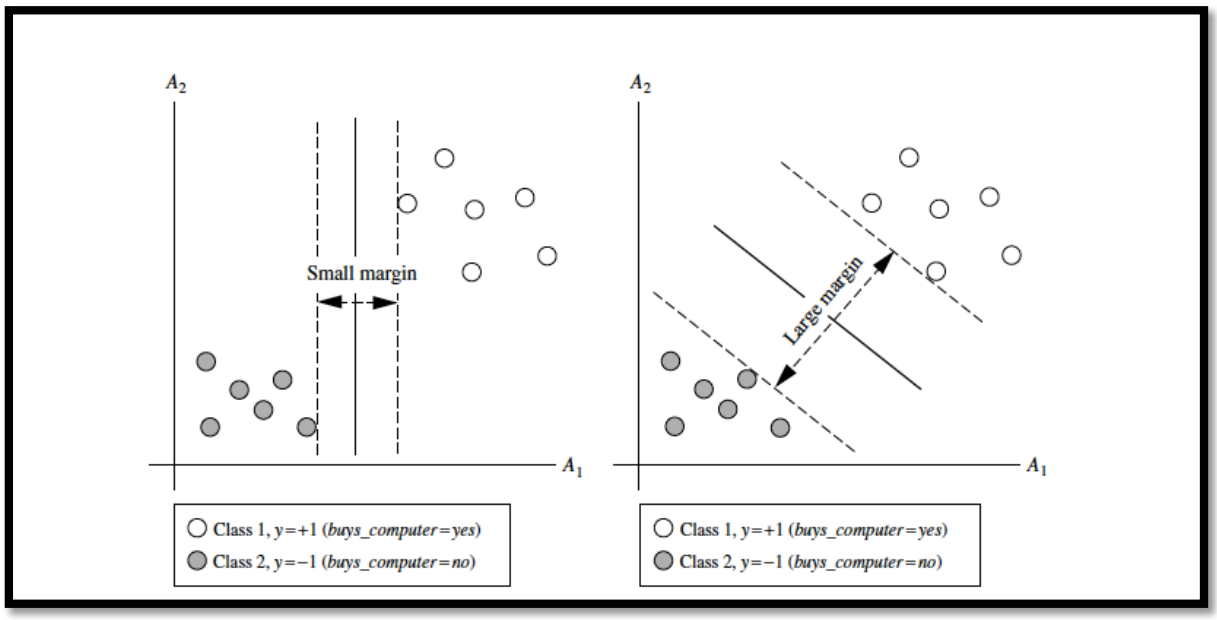

Fig 3.4: Linearly separable data: small margin vs large margin

When the data is linearly inseparable, a nonlinear SVM is constructed by extending the approach for linear SVMs as follows. There are two main steps. In the first step, transformation of the original input data into a higher dimensional space using a nonlinear mapping is done (Fig 3.5). Once the data have been transformed into the new higher space, the second step searches for a linear separating hyperplane in the new space. The maximal marginal hyperplane found in the new space corresponds to a nonlinear separating hypersurface in the original space.

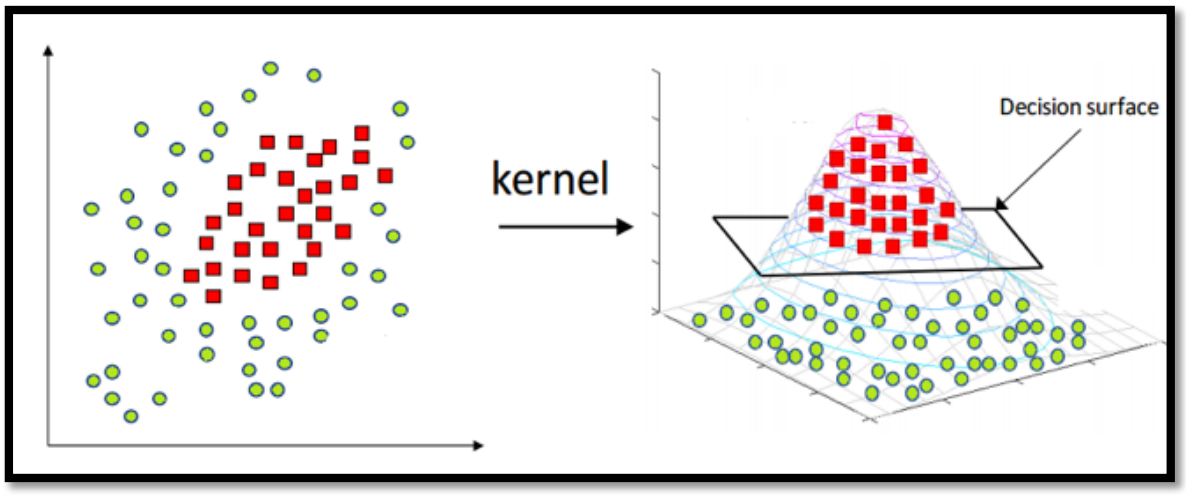

Fig 3.5: A simple 2-D case of linearly inseparable data transformed using kernel function

The kernel used for classification is 'RBF kernel' (Radial Basis Function kernel; also known as Gaussian Kernel). It was observed that RBF kernel performed better compared to linear and polynomial kernels.

$$\textbf{Gaussian radial basis function kernel:} \quad K(X_i, X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$$

where $K(\mathbf{X_i}, \mathbf{X_j}) = \varphi(X_i) \cdot \varphi(X_j)$ is the kernel function. the training tuples appear only in the form of dot products, $\varphi(X_i) \cdot \varphi(X_j)$, where $\varphi(X)$ is simply the nonlinear mapping function applied to transform the training tuples. In other words, everywhere that $\varphi(X_i) \cdot \varphi(Xj)$ appears in the training algorithm, we can replace it with $K(X_i, X_j)$. The fitcecoc()[10] method is used to fit an SVM model to the given model. This is the only method which classifies data into multiple classes.

Decision tree induction is the learning of decision trees from class-labelled training tuples. A decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node holds a class label. The topmost node in a tree is the root node[8]. Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle multidimensional data. Their representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans. The learning and classification steps of decision tree induction are simple and fast. In general, decision tree classifiers have good accuracy. However, successful use may depend on the data at hand. Decision tree induction algorithms have been used for classification in many application areas such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology. The function fitctree()[11] is used to fit binary classification decision tree for multiclass classification.

A classification ensemble is a predictive model composed of a weighted combination of multiple classification models (Fig 3.6). In general, combining multiple classification models increases predictive performance.

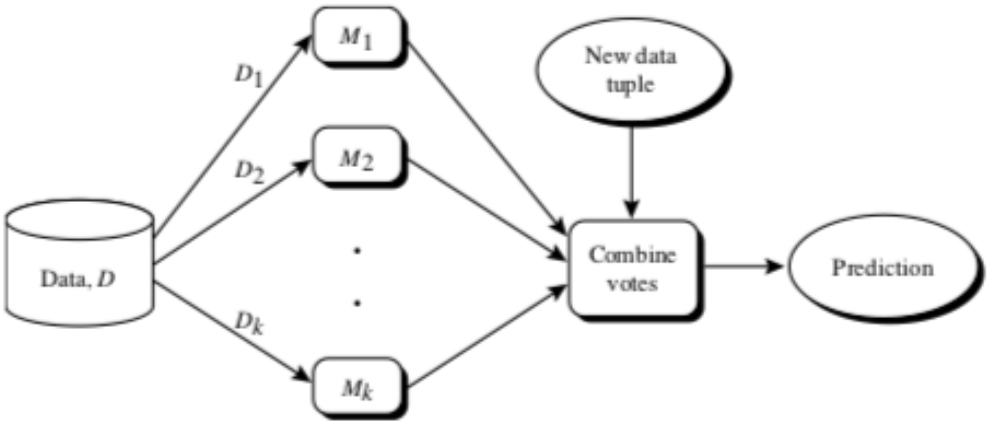

Fig 3.6: Ensemble methods for increasing classifier accuracy

Bagging, which stands for "bootstrap aggregation," is a type of ensemble learning. Bagging works by training learners on resampled versions of the data. This resampling is usually done by bootstrapping observations, that is, selecting N out of N observations with replacement for every new learner. In addition, every tree in the ensemble can randomly select predictors for decision splits—a technique known to improve the accuracy of bagged trees[12]. Given a set, D, of d tuples, bagging works as follows. For iteration i (i = 1, 2, . . . , k), a training set, Di, of d tuples is sampled with replacement from the original set of tuples, D. Note that the term bagging stands for bootstrap aggregation. Each training set is a bootstrap sample. Since sampling with replacement is used, some of the original tuples of D may not be included in D_i , whereas others may occur more than once. A classifier model, M_i, is learned for each training set, D_i. To classify an unknown tuple, X, each classifier, M_i, returns its class prediction, which counts as one vote. The bagged classifier, M∗, counts the votes and assigns the class with the most votes to X. The bagged classifier often has significantly greater accuracy than a single classifier derived from D, the original training data (Fig 3.8). It will not be considerably worse and is more robust to the effects of noisy data and overfitting (Fig 3.7). The increased accuracy occurs because the composite model reduces the variance of the individual classifiers.

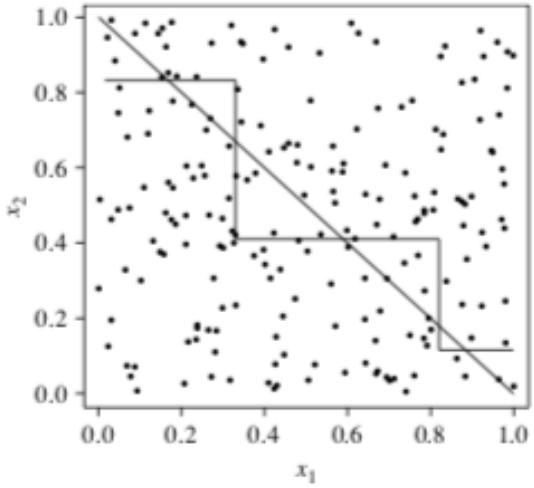

Fig 3.7: Decision boundary by a single decision tree for a linearly separable problem

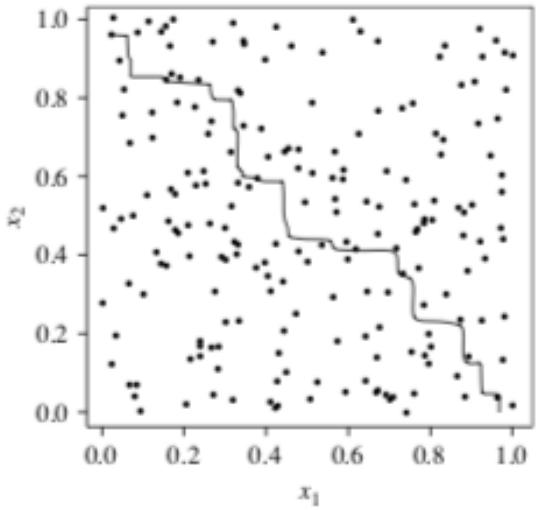

Fig 3.8: Decision boundary by an ensemble of decision trees for a linearly separable problem

fitensemble()[13] is used to train ensemble method. fitensemble can boost or bag decision tree learners or discriminant analysis classifiers. The function can also train random subspace ensembles of KNN or discriminant analysis classifiers. The ensemble model trained here is a combination of 100 models (i.e., 100 learning cycles). All the sub-models trained under ensembles are tree classifiers.

A convolutional neural network (CNN or ConvNet) is a type of machine learning algorithm in which a model learns to perform classification tasks directly from images, video, text, or sound[14]. CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They learn directly from image data, using patterns to classify images and eliminating the need for manual feature extraction. A convolution layer, batch normalization

layer, relu layer, max pooling layer, average pooling layer, fully connected layer and a soft-max layer are used to train the model. Adam's optimization technique has been utilized along with other optimizing parameters such as initial learning rate, L2 regularization, max epochs, etc. to get the best tuned model. Bayesian Hyper parameter tuning technique is used to atone the best possible configuration of these features[15].

3.2.4 Performance Analysis Module

The performance of all the models built is analysed using k-fold cross validation. In k-fold cross-validation, the initial data are randomly partitioned into k mutually exclusive subsets or "folds," D_1, D_2,..., D_k, each of approximately equal size. Training and testing is performed k times. In iteration i, partition D_i is reserved as the test set, and the remaining partitions are collectively used to train the model. That is, in the first iteration, subsets D_2,..., D_k collectively serve as the training set to obtain a first model, which is tested on D_1; the second iteration is trained on subsets D_1, D_3,..., D_k and tested on D_2; and so on. Unlike the holdout and random subsampling methods, here each sample is used the same number of times for training and once for testing. For classification, the accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of tuples in the initial data. Leave-one-out is a special case of k-fold cross-validation where k is set to the number of initial tuples. That is, only one sample is "left out" at a time for the test set. In stratified cross-validation, the folds are stratified so that the class distribution of the tuples in each fold is approximately the same as that in the initial data.

In this project for SVM, decision tree and ensembles, the k value of k-fold cross validation was was chosen as 10. For CNN, the training, validation and test sets are split in the ratio of 60:20:20.

4. Testing and Results

The models that were built were tested with various to get the best possible set of results. This process included manually giving various parameters to models such as SVM, decision trees and ensembles (bagging). For CNN, this process was automated with the help of Bayesian Hyper Parameter Tuning technique to get the best set of possible values for parameters, which give the best convolutional neural network model. The test cases and results for all the models are given below:

4.1 Support Vector Machine Testing and Results:

SVM with linear kernel code is as shown in Fig 4.1 and its results are as shown in Fig 4.2:

```
%SVM
%hyperparameters = struct('kernal_scale', [1e-3, 1e3]);
svmParams = templateSVM('KernelFunction','linear', 'KernelScale', 1, ...
    'Standardize', true, 'BoxConstraint', 2, 'Cost', [0, 1; 10, 0]);
%minfn = @(z)kfoldLoss(fitcecoc(data, label,'Learners', svmParams, 'Coding', 'onevsall'));
%result = bayesopt(minfn,kernal_scale)
svm_Mdl = fitcecoc(data, label, 'Learners', svmParams, 'Coding', 'onevsall');
svm_CVMdl = crossval(svm_Mdl);
svm_loss = kfoldLoss(svm_CVMdl, 'LossFun', 'classiferror')
```

Fig 4.1: Code of SVM with linear kernel

```
svm_loss =

    0.4370
```

Fig 4.2: Loss of SVM with linear kernel

SVM with polynomial kernel code is as shown in Fig 4.3 and its results are as shown in Fig 4.4:

```
%SVM
%hyperparameters = struct('kernal_scale', [1e-3, 1e3]);
svmParams = templateSVM('KernelFunction','polynomial', 'KernelScale', 1, ...
    'Standardize', true, 'BoxConstraint', 2, 'Cost', [0, 1; 10, 0]);
%minfn = @(z)kfoldLoss(fitcecoc(data, label,'Learners', svmParams, 'Coding', 'onevsall'));
%result = bayesopt(minfn,kernal_scale)
svm_Mdl = fitcecoc(data, label, 'Learners', svmParams, 'Coding', 'onevsall');
svm_CVMdl = crossval(svm_Mdl);
svm_loss = kfoldLoss(svm_CVMdl, 'LossFun', 'classiferror')
```

Fig 4.3 : Code of SVM with polynomial kernel

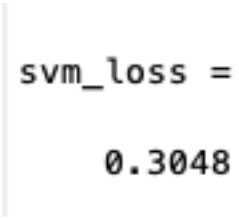

Fig 4.4: Loss of SVM with Polynomial kernel

SVM with RBF kernel code is as shown in Fig 4.5 and its results are as shown in Fig 4.6:

```
%SVM
%hyperparameters = struct('kernal_scale', [1e-3, 1e3]);
svmParams = templateSVM('KernelFunction','rbf', 'KernelScale', 1, ...
    'Standardize', true, 'BoxConstraint', 2, 'Cost', [0, 1; 10, 0]);
%minfn = @(z)kfoldLoss(fitcecoc(data, label,'Learners', svmParams, 'Coding', 'onevsall'));
%result = bayesopt(minfn,kernal_scale)
svm_Mdl = fitcecoc(data, label, 'Learners', svmParams, 'Coding', 'onevsall');
svm_CVMdl = crossval(svm_Mdl);
svm_loss = kfoldLoss(svm_CVMdl, 'LossFun', 'classiferror')
```

Fig 4.5: Code of SVM with RBF kernel

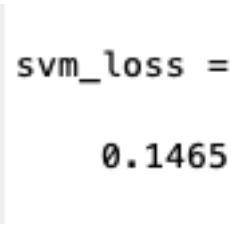

Fig 4.6: Loss of SVM with RBF kernel

Thus, it was concluded that **RBF kernel gave the best results** (least loss of 0.1465) with the least time consumed. SVM with polynomial kernel performed the second best. However, it consumed more time compared to linear kernel. Linear kernel gave a loss 0.4370, which is very high and is not fit classification.

4.2 Decision Tree Results:

The code of Binary Decision Tree is as shown in Fig 4.7:

```
%Binary Decision Tree
tree_Mdl = fitctree(data,label);
tree_CVMdl = crossval(tree_Mdl);
tree_loss = kfoldLoss(tree_CVMdl, 'LossFun', 'classiferror')
```

Fig 4.7: Code of Decision Tree

Loss obtained for Decision Tree as shown in Fig 4.8:

```
tree_loss =

    0.1958
```

Fig 4.8: Loss obtained for Decision Tree

4.3 Ensemble Methods Testing and Results:

The code of ensemble methods is as shown in Fig 4.9:

```
%Ensembles
ens_Mdl = fitensemble(data, label, 'Bag', 100, 'Tree', 'Type', 'Classification');
ens_CVMdl = crossval(ens_Mdl);

ens_loss = kfoldLoss(ens_CVMdl,'Mode','Cumulative')
figure, plot(ens_loss), title('Ensembles');
xlabel('Number of Iterations');
ylabel('Classification Loss');
```

Fig 4.9: Code of Ensemble Method (Bagging)

Loss obtained for Ensemble method (Bagging) as shown in Fig 4.10 and Fig 4.11:

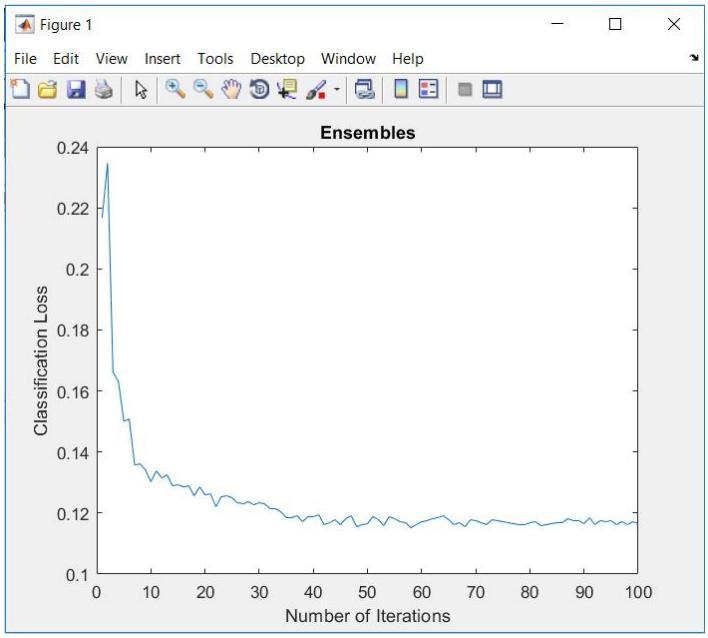

Fig 4.10: Variation of classification loss with the number of models constructed (Bagging)

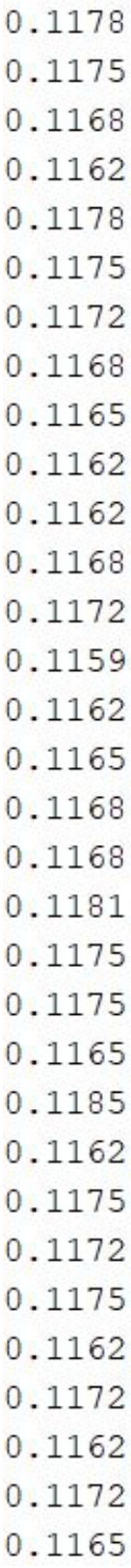

Fig 4.11: A snippet of classification loss for the various models constructed under Bagging

```
%Ensembles
ens_Mdl = fitensemble(data, label, 'AdaBoostM1', 100, 'Tree', 'Type', 'Classification');
ens_CVMdl = crossval(ens_Mdl);

ens_loss = kfoldLoss(ens_CVMdl,'Mode','Cumulative')
figure, plot(ens_loss), title('Ensembles');
xlabel('Number of Iterations');
```

Fig 4.12: Code of Ensemble Method (AdaBoostM1)

```
Error using classreg.learning.modelparams.EnsembleParams/fillDefaultParams (line 529)
You cannot use method AdaBoostM1 with more than 2 classes.
```

Fig 4.13: Error message for using AdaBoostM1 for more that 2 classes

```
%Ensembles
ens_Mdl = fitensemble(data, label, 'AdaBoostM2', 100, 'Tree', 'Type', 'Classification');
ens_CVMdl = crossval(ens_Mdl);

ens_loss = kfoldLoss(ens_CVMdl,'Mode','Cumulative')
figure, plot(ens_loss), title('Ensembles');
xlabel('Number of Iterations');
```

Fig 4.14: Code of Ensemble Method (AdaBoostM2)

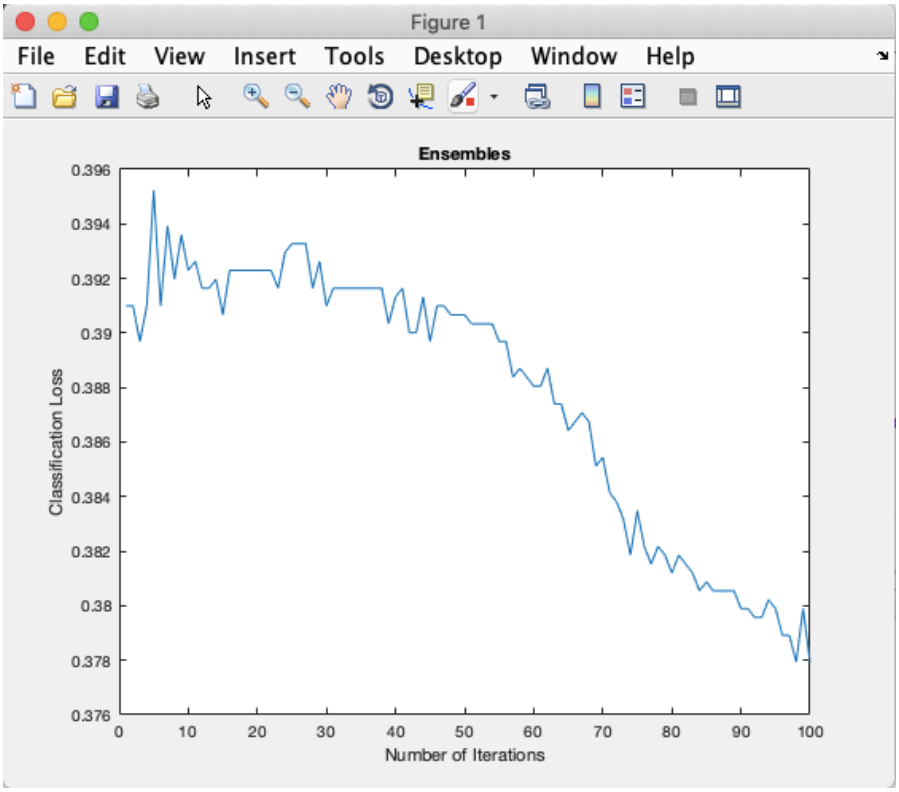

Fig 4.15: Variation of classification loss with the number of models constructed (AdaBoostM2)

```
%Ensembles
ens_Mdl = fitensemble(data, label, 'TotalBoost', 100, 'Tree', 'Type', 'Classification');
ens_CVMdl = crossval(ens_Mdl);

ens_loss = kfoldLoss(ens_CVMdl,'Mode','Cumulative')
figure, plot(ens_loss), title('Ensembles');
xlabel('Number of Iterations');
ylabel('Classification Loss');
```

Fig 4.16: Code of Ensemble Method (TotalBoost)

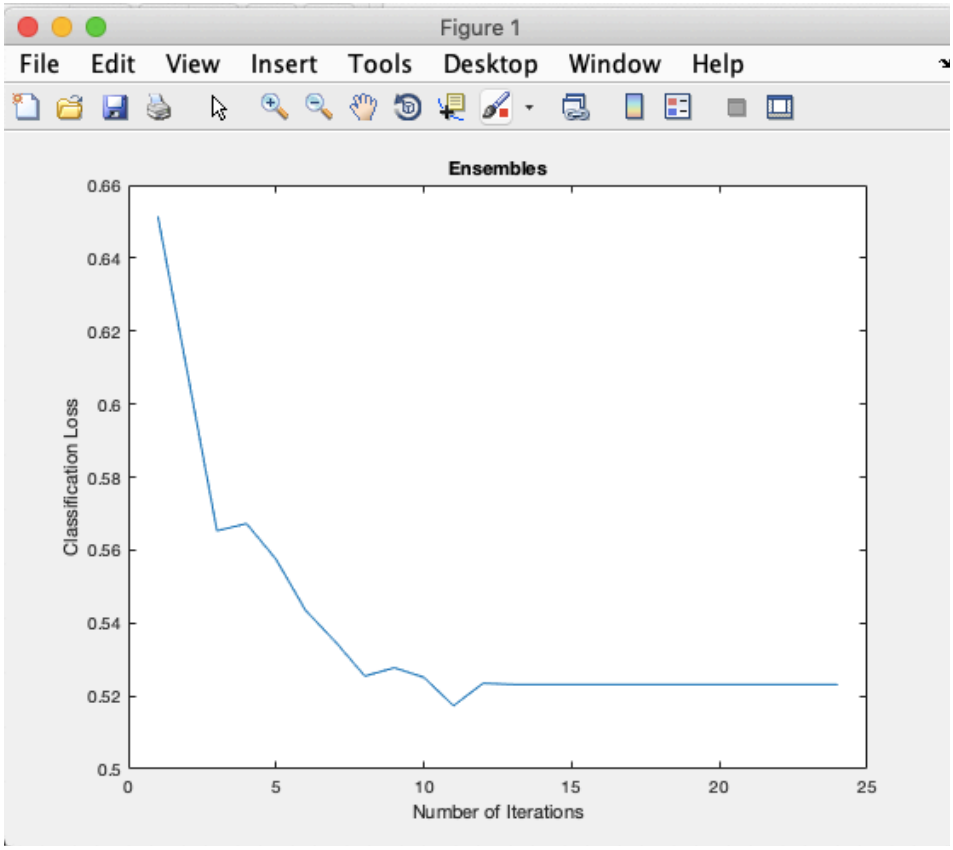

Fig 4.17: Variation of classification loss with the number of models constructed (TotalBoost)

Thus, on the basis of the above observations, it is concluded that **Bagging method performed considerably better** than other ensemble methods. AdaBoostM1 failed to classify as it works only for 2 class implementation (Fig 4.12 and Fig 4.13). AdaBoostM2 (Fig 4.14 and Fig 4.15), TotalBoost (Fig 4.16 and Fig 4.17) and other methods gave worse results than Bagging method. In comparison, decision tree (Fig 4.7 and Fig 4.8) gave an accuracy of just 81.42%.

4.4 Convolutional Neural Network (CNN) Testing and Results:

When using CNN, one has the option to automate the process of testing various hyper parameters, rather than manual testing. Bayesian Optimization algorithm was used for this process (Fig 4.17).

The Bayesian optimization algorithm attempts to minimize a scalar objective function f(x) for x in a bounded domain. The function can be deterministic or stochastic, meaning it can return different results when evaluated at the same point x. The components of x can be continuous reals, integers, or categorical, meaning a discrete set of names.

The key elements in the minimization are:

- A Gaussian process model of f(x).
- A Bayesian update procedure for modifying the Gaussian process model at each new evaluation of f(x).
- An acquisition function a(x) (based on the Gaussian process model of f) that you maximize to determine the next point x for evaluation.

Algorithm outline:

- Evaluate $y_i = f(x_i)$ for NumSeedPoints points x_i, taken at random within the variable bounds. NumSeedPoints is a bayesopt setting. If there are evaluation errors, take more random points until there are NumSeedPoints successful evaluations. The probability distribution of each component is either uniform or log-scaled, depending on the Transform value in optimizableVariable.

Then repeat the following steps:

- Update the Gaussian process model of f(x) to obtain a posterior distribution over functions Q(f|xi, yi for i = 1,...,t).
- Find the new point x that maximizes the acquisition function a(x).

The algorithm stops after reaching any of the following:

- A fixed number of iterations (default 30).
- A fixed time (default is no time limit).

- A stopping criterion that you supply in Bayesian Optimization Output Functions or Bayesian Optimization Plot Functions.

The function bayesopt()[14] was used to select optimal machine learning hyper parameters using Bayesian optimization (Fig 4.18 and Fig 4.19).

```
optimVars = [
    optimizableVariable('InitialLearnRate',[0.001 0.005],'Transform','log');
    optimizableVariable('GradientDecayFactor',[0.8 0.95], 'Transform','log');
    optimizableVariable('SquaredGradientDecayFactor',[0.85 1],'Transform','log');
    optimizableVariable('L2Regularization',[1e-10 1e-2],'Transform','log');
    optimizableVariable('MaxEpochs',[16 32],'Type','integer')];

i = 1;
ObjFcn = makeObjFcn(imdsTrain,imdsValidation,datasource,imageSize,i);
BayesObject = bayesopt(ObjFcn,optimVars,...
    'MaxObj',30,...
    'MaxTime',8*60*60,...
    'IsObjectiveDeterministic',false,...
    'UseParallel',false);


bestIdx = BayesObject.IndexOfMinimumTrace(end);
fileName = BayesObject.UserDataTrace{bestIdx};
savedStruct = load(fileName);
valError = savedStruct.valError
```

Fig 4.18: Code for performing Bayesian optimization

Command Window

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NetworkDepth	InitialLearn-Rate	Momentum	L2Regulariza-tion
1	Best	0.30903	237.73	0.30903	0.30903	1	0.0033168	0.81662	0.0058601
2	Accept	0.44723	398.32	0.30903	0.31592	3	0.032427	0.80096	1.7683e-08
3	Best	0.2753	397.07	0.2753	0.28314	3	0.0033159	0.88593	3.0228e-05
4	Accept	0.31665	236.85	0.2753	0.27532	1	0.0044057	0.84082	3.9617e-10
5	Accept	0.30141	397.58	0.2753	0.27532	3	0.0011607	0.9499	0.0015308
6	Accept	0.36344	396.29	0.2753	0.27531	3	0.0024446	0.94716	1.1102e-07
7	Accept	0.47008	395.85	0.2753	0.29967	3	0.01203	0.94174	0.00016319
8	Accept	0.30903	235.79	0.2753	0.30073	1	0.0036399	0.94546	6.4899e-07
9	Best	0.2111	235.54	0.2111	0.21112	1	0.0010023	0.94945	0.0052037
10	Accept	0.28183	235.65	0.2111	0.21113	1	0.0011609	0.94268	1.0224e-06
11	Accept	0.41567	240.98	0.2111	0.21113	1	0.09987	0.93803	0.0038623
12	Accept	0.26986	319.86	0.2111	0.26477	2	0.0010008	0.86936	1.3023e-06
13	Accept	0.24701	238.74	0.2111	0.25097	1	0.0010002	0.94927	2.1208e-10
14	Accept	0.23939	235.9	0.2111	0.241	1	0.0010029	0.94698	9.0976e-07
15	Best	0.20675	235.78	0.20675	0.23416	1	0.0010004	0.9185	4.5898e-07
16	Accept	0.3852	236.72	0.20675	0.23371	1	0.056459	0.82482	4.537e-08
17	Accept	0.38194	236.9	0.20675	0.24274	1	0.019722	0.90275	2.474e-08
18	Accept	0.36017	398.21	0.20675	0.23184	3	0.099739	0.81094	7.32e-06
19	Accept	0.26877	236.45	0.20675	0.2317	1	0.0010049	0.84187	4.6356e-05
20	Best	0.19042	239.44	0.19042	0.20403	1	0.0010002	0.93503	0.0022395

Fig 4.19: Number of models being built under Bayesian Optimization

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 8750.4271 seconds.
Total objective function evaluation time: 8724.989

Best observed feasible point:
    NetworkDepth    InitialLearnRate    Momentum    L2Regularization
    ____________    ________________    ________    ________________

        1             0.0010002         0.93503        0.0022395

Observed objective function value = 0.19042
Estimated objective function value = 0.24057
Function evaluation time = 239.4382

Best estimated feasible point (according to models):
    NetworkDepth    InitialLearnRate    Momentum    L2Regularization
    ____________    ________________    ________    ________________

        1             0.0010163         0.91039        0.00084603

Estimated objective function value = 0.24057
Estimated function evaluation time = 236.8533
```

Fig 4.20: Optimal Hyper Parameters obtained after the completion of Hyper parameter tuning

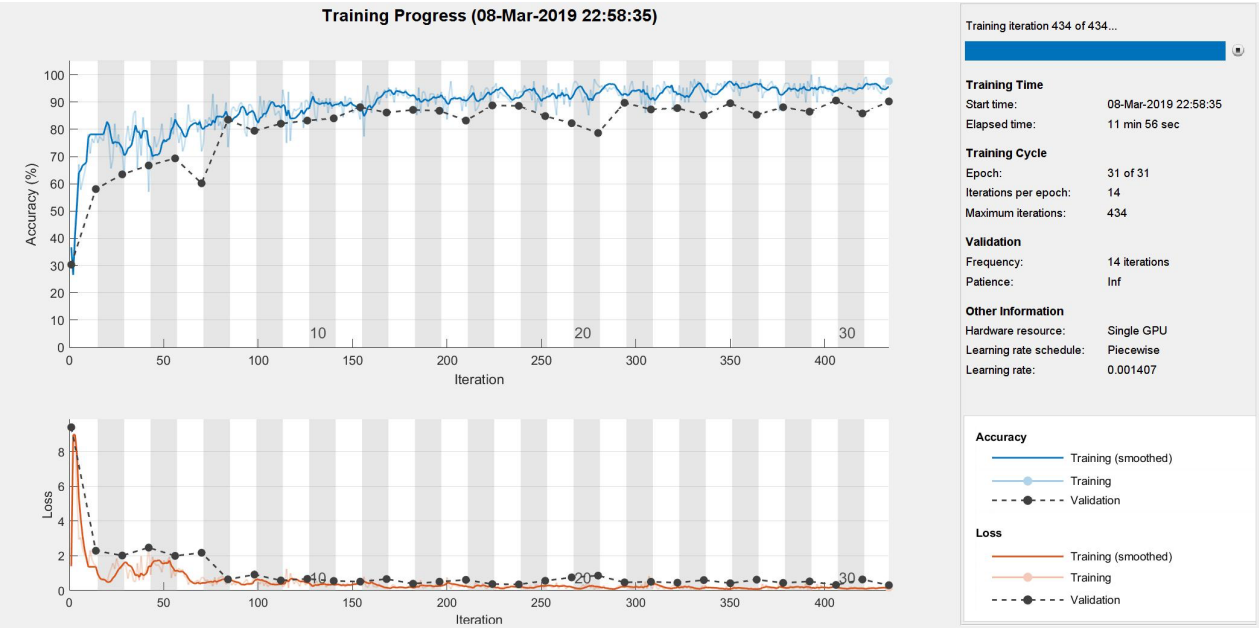

Fig 4.21: Training progress with optimal hyper parameters

After the completion of hyper parameter tuning, the model trained with the optimal hyper parameters resulted in an accuracy of 90.21% on validation set (Fig 4.20).

5. Conclusions and Future Scope

The process of detecting a tumour in brain was carried using various modern machine learning algorithms such as Support Vector Machine (SVM), Ensemble methods and Convolutional Neural Network (CNN). The entire training was carried on a dataset of 3064 T1 weighted MR images. After the evaluation of results obtained from various models, it was observed that CNN gave the best results among these models with an accuracy of 90.21% on validation set. Ensemble method – Bagging gave an accuracy of 88.41% on validation set. Support Vector Machine gave an accuracy of 85.35% on validation set. Additionally, dataset was augmented with additional images by rotating, translating and cropping the existing images. With the help of GPU, the computation was speeded-up in the case of CNN training and huge amount of time was saved. Using RBF kernel in SVM gave the best results when compared to linear and polynomial kernels. In Ensemble methods, Bagging for tree models gave the best results compared to AdaBoostM2, TotalBoost etc.

Future Scope

Incorporation of Image Processing techniques such as image segmentation edge detection and morphological operations can lead to increase in the accuracies with which the models can identify the tumour. Increasing the dataset can be very helpful as it increases the training set size and the model becomes more accurate.

References

[1] Nilesh Bhaskarrao Bahadure, Arun Kumar Ray, and Har Pal Thethi, "Image Analysis for MRI Based Brain Tumour Detection and Feature Extraction Using Biologically Inspired BWT and SVM," International Journal of Biomedical Imaging, vol. 2017, Article ID 9749108, 12 pages, 2017, pp 155-158

[2] Manu BN. (2016, Feb.). "Brain MRI Tumour Detection and Classification". [Online]. Available: https://in.mathworks.com/matlabcentral/fileexchange/55107-brain-mri-tumor-detection-and-classification

[3] Chandra Sekhar Ravuri. (2015, Jun.). "Automatic segmentation of brain tumour in MR images". [Online]. Available: https://in.mathworks.com/matlabcentral/fileexchange/51153-automatic-segmentation-of-brain-tumor-in-mr-images

[4] Cheng J, Huang W, Cao S, Yang R, Yang W, Yun Z, et al. "Enhanced Performance of Brain Tumour Classification via Tumour Region Augmentation and Partition", 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0140381

[5] Madhumantee Naskar, "Automated System for Brain Tumour Detection and Segmentation", Journal of Engineering Research and Studies,2018, vol. VI, issue I, pp. 03-05

[6] Ari, A., & Hanbay, D. (2018). "Deep learning based brain tumour classification and detection system". Turkish Journal of Electrical Engineering & Computer Sciences, Vol.26, No.5, pp.2275-2286.

[7] J. Han, M. Kamber and J. Pei, "Data mining". Waltham, MA: Morgan Kaufmann/Elsevier, 2012, ch. 4, pp. 100-102

[8] T. Mitchell. "Machine learning". New York: McGraw-Hill, 2013, ch. 3, p. 56

[9] Ajay Ungar. (2017, Mar.). IIT Roorkee, India. [Online]. Available: https://medium.com/data-science-group-iitr/support-vector-machines-svm-unraveled-e0e7e3ccd49b

[10] MathWorks. (2014, Aug.). Natick, MA, USA. [Online]. Available: https://in.mathworks.com/help/stats/fitcecoc.html?s_tid=doc_ta

[11] MathWorks. (2014, Jan.). Natick, MA, USA. [Online]. Available: https://in.mathworks.com/help/stats/fitctree.html?s_tid=doc_ta

[12] Dave Sotelo. (2017, Jul.). San Francisco, CA, USA. [Online]. Available: https://towardsdatascience.com/using-bagging-and-boosting-to-improve-classification-tree-accuracy-6d3bb6c95e5b

[13] MathWorks. (2011, Jan.). Natick, MA, USA. [Online]. Available: https://in.mathworks.com/help/stats/fitensemble.html?s_tid=doc_ta

[14] MathWorks. (2016, Jul.). Natick, MA, USA. [Online]. Available: https://in.mathworks.com/help/stats/bayesopt.html

[15] MathWorks. (2016, Jul.). Natick, MA, USA. [Online]. Available: https://in.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Appendix – A: User Manual for Effective Brain Tumour Detection using Modern ML Approaches

Step 1: Create an account with MathWorks and sign-in.

Step 2: Download MATLAB from www.mathworks.com.

Step 3: Install MATLAB on your local system by providing the licence key.

Step 4: Navigate to the folder containing the code for brain tumour detection.

Step 5: Run 'FeatureExtraction.m' by clicking on 'Run' button on the toolbar. Features will be extracted and 'Trainset.mat' will be generated in data folder, which contains the extracted features of the entire data.

Step 6: Run 'Training.m' by clicking on 'Run' button on the toolbar. The classification is performed here using SVM , decision trees and bagging. The results are displayed on the console.

Step 7: Run 'cnn.m' by clicking on 'Run' button on the toolbar. Multiple models get trained and the test results are displayed on the console for the best model trained.

Appendix – B: Sample Code

FeatureExtraction.m

```
%This script is used to extract features from the data
clear;
close;
clc;

data = [];
label = [];
corrupt_images = [];
for k = 1:3064
    load(strcat('../Data/',num2str(k),'.mat'));
    img = cjdata.image;
    seg_img = uint8(255*mat2gray(img));

    s = size(seg_img);
    if ((s(1) ~= 512) || (s(2) ~= 512))
        seg_img = imresize(seg_img, 2);
        img = imresize(img, 2);
    end
    l = cjdata.label;
    % Extract features using DWT
    x = double(seg_img);
    m = size(seg_img,1);
    n = size(seg_img,2);
    %signal1 = (rand(m,1));
    %winsize = floor(size(x,1));
    %winsize = int32(floor(size(x)));
    %wininc = int32(10);
    %J = int32(floor(log(size(x,1))/log(2)));
    %Features = getmswpfeat(signal,winsize,wininc,J,'matlab');
```

```
%m = size(img,1);
%signal = rand(m,1);
signal1 = seg_img(:,:);
%Feat = getmswpfeat(signal,winsize,wininc,J,'matlab');
%Features = getmswpfeat(signal,winsize,wininc,J,'matlab');
%Using DWT
[cA1,cH1,cV1,cD1] = dwt2(signal1,'db4');
[cA2,cH2,cV2,cD2] = dwt2(cA1,'db4');
[cA3,cH3,cV3,cD3] = dwt2(cA2,'db4');

DWT_feat = [cA3,cH3,cV3,cD3];
G = pca(DWT_feat);
%whos DWT_feat
%whos G
g = graycomatrix(G);
stats = graycoprops(g,'Contrast Correlation Energy Homogeneity');
Contrast = stats.Contrast;
Correlation = stats.Correlation;
Energy = stats.Energy;
Homogeneity = stats.Homogeneity;
Mean = mean2(G);
Standard_Deviation = std2(G);
Entropy = entropy(G);
RMS = mean2(rms(G));
%Skewness = skewness(img)
Variance = mean2(var(double(G)));
a = sum(double(G(:)));
Smoothness = 1-(1/(1+a));
Kurtosis = kurtosis(double(G(:)));
Skewness = skewness(double(G(:)));
% Inverse Difference Movement
m = size(G,1);
n = size(G,2);
in_diff = 0;
```

```
    for i = 1:m
        for j = 1:n
            temp = G(i,j)./(1+(i-j).^2);
            in_diff = in_diff+temp;
        end
    end
    IDM = double(in_diff);
%Aggregating the features and data
    feat = [Contrast,Correlation,Energy,Homogeneity, Mean, Standard_Deviation, Entropy,
RMS, Variance, Smoothness, Kurtosis, Skewness, IDM, l];
    data=[data;feat];
    k
end
save('..\Data\Trainset.mat','data'); %Saved as 'Trainset.mat'
```

Training.m

```
%This script is used to train SVM, decision tree and ensemble methods(bagging)
clear all;
clc;
close all;
seed = 100;
rng(seed);

%Loading the data and labels

load('../Data/Trainset.mat');
data = datasample(data,3064);
label = data(:,end);
data = data(:,1:end-1);

%SVM
%hyperparameters = struct('kernal_scale', [1e-3, 1e3]);
svmParams = templateSVM('KernelFunction','rbf', 'KernelScale', 1, ...
   'Standardize', true, 'BoxConstraint', 2, 'Cost', [0, 1; 10, 0]);
%minfn = @(z)kfoldLoss(fitcecoc(data, label,'Learners', svmParams, 'Coding', 'onevsall'));
%result = bayesopt(minfn,kernal_scale)
svm_Mdl = fitcecoc(data, label, 'Learners', svmParams, 'Coding', 'onevsall');
svm_CVMdl = crossval(svm_Mdl);
svm_loss = kfoldLoss(svm_CVMdl, 'LossFun', 'classiferror')

%Binary Decision Tree
tree_Mdl = fitctree(data,label);
tree_CVMdl = crossval(tree_Mdl);
%Calculating kfoldLoss
tree_loss = kfoldLoss(tree_CVMdl, 'LossFun', 'classiferror')
```

```
%KNN
knn_Mdl = fitcknn(data, label);
knn_CVMdl = crossval(knn_Mdl);
%Calculating kfoldLoss
knn_loss = kfoldLoss(knn_CVMdl, 'LossFun', 'classiferror')

%Ensembles
ens_Mdl = fitensemble(data, label, 'Subspace', 100, 'Tree', 'Type', 'Classification');
ens_CVMdl = crossval(ens_Mdl);

ens_loss = kfoldLoss(ens_CVMdl,'Mode','Cumulative')
%Plotting the ensembles
figure, plot(ens_loss), title('Ensembles');
xlabel('Number of Iterations');
ylabel('Classification Loss');
```

cnn.m

```
%This script is used to train various CNN models for classification of tumour images
clear all;

clc;

close all;

a=rng;
out=randn;
rng(a)
out=randn;
 %Setting the folder
rootFolder = '../Data_img/';

imds = imageDatastore(rootFolder, ...
   'IncludeSubfolders',true, ...
   'LabelSource','foldernames');
%Splitting the the dataset into Training, Validation and Test Data
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.6);
[imdsValidation, imdsTest] = splitEachLabel(imdsValidation, 0.5);

%Augmenting the data
pixelRange = [-4 4];
rotationRange = [0 90];
imageAugmenter = imageDataAugmenter(...
        'RandRotation',rotationRange, ...
        'RandXReflection',true,...
        'RandYReflection',true,...
        'RandXTranslation',pixelRange,...
        'RandYTranslation',pixelRange);
imageSize = [128 128 1];
datasource = augmentedImageDatastore(imageSize,imdsTrain,...
```

```
        'DataAugmentation',imageAugmenter,...
        'OutputSizeMode','randcrop');

%Setting the range of Hyper parameters to be tuned
optimVars = [
   optimizableVariable('InitialLearnRate',[0.001 0.005],'Transform','log');
   optimizableVariable('GradientDecayFactor',[0.8 0.95], 'Transform','log');
   optimizableVariable('SquaredGradientDecayFactor',[0.85 1],'Transform','log');
   optimizableVariable('L2Regularization',[1e-10 1e-2],'Transform','log');
   optimizableVariable('MaxEpochs',[16 32],'Type','integer')];

i = 1;
%Setting the Bayesian Function Parameters
ObjFcn = makeObjFcn(imdsTrain,imdsValidation,datasource,imageSize,i);
BayesObject = bayesopt(ObjFcn,optimVars,...
   'MaxObj',30,...
   'MaxTime',8*60*60,...
   'IsObjectiveDeterministic',false,...
   'UseParallel',false);



bestIdx = BayesObject.IndexOfMinimumTrace(end);
fileName = BayesObject.UserDataTrace{bestIdx};
savedStruct = load(fileName);
valError = savedStruct.valError

[YPredicted,probs] = classify(savedStruct.trainedNet,imdsTest);
YTest = imdsTest.Labels;
testError = 1 - mean(YPredicted == YTest)
 %Calculation of test error
NTest = numel(YTest);
testErrorSE = sqrt(testError*(1-testError)/NTest);
testError95CI = [testError - 1.96*testErrorSE, testError + 1.96*testErrorSE]
```

```
figure
[cmat,classNames] = confusionmat(YTest,YPredicted);
h = heatmap(classNames,classNames,cmat);
%Plotting the data
xlabel('Predicted Class');
ylabel('True Class');
title('Confusion Matrix');
```

makeObjFcn.m

```
%Defining the Objective Function is done in this script
function ObjFcn = makeObjFcn(imdsTrain,imdsValidation,datasource,imageSize,i)
ObjFcn = @valErrorFun;
    function [valError,cons,fileName] = valErrorFun(optVars)
        numClasses = numel(unique(imdsTrain.Labels));
        initialNumFilters = 16;

%Defining the layers of CNN

        layers = [
            imageInputLayer(imageSize)

            convBlock(3,initialNumFilters)
            maxPooling2dLayer(2,'Stride',2)

            convBlock(3,2*initialNumFilters)
            maxPooling2dLayer(2,'Stride',2)

            convBlock(3,4*initialNumFilters)
            maxPooling2dLayer(8)

            convBlock(3,16*initialNumFilters)
            averagePooling2dLayer(8)

            fullyConnectedLayer(numClasses)
            softmaxLayer
            classificationLayer];

        miniBatchSize = 128;
        validationFrequency = floor(numel(imdsTrain.Files)/miniBatchSize);
        options = trainingOptions('adam',...
            'InitialLearnRate',optVars.InitialLearnRate,...
```

```
        'GradientDecayFactor',optVars.GradientDecayFactor, ...
        'SquaredGradientDecayFactor',optVars.SquaredGradientDecayFactor, ...
        'MaxEpochs',optVars.MaxEpochs, ...
        'LearnRateSchedule','piecewise',...
        'LearnRateDropPeriod',35,...
        'LearnRateDropFactor',0.1,...
        'MiniBatchSize',miniBatchSize,...
        'L2Regularization',optVars.L2Regularization,...
        'Shuffle','every-epoch',...
        'Verbose',false,...
        'Plots','training-progress',...
        'ValidationData',imdsValidation,...
        'ValidationPatience',Inf,...
        'ValidationFrequency',validationFrequency,...
        'OutputFcn',@(info)savetrainingplot(info, i));

    trainedNet = trainNetwork(datasource,layers,options);

    close(findall(groot,'Tag','NNET_CNN_TRAININGPLOT_FIGURE'))
%Saving the trained models
    rootfile = '../Network Progress - 4/';
    YPredicted = classify(trainedNet,imdsValidation);
    valError = 1 - mean(YPredicted == imdsValidation.Labels);
    fileName = strcat(num2str(valError),'-',num2str(i),'.mat');
    save(strcat(rootfile,'Trained Nets/',fileName),'trainedNet','valError','options')
    cons = [];
    i=i+1;
  end
end

function layers = convBlock(filterSize,numFilters)
layers = [
  convolution2dLayer(filterSize,numFilters,'Padding','same')
  batchNormalizationLayer
```

```
    reluLayer];
end


function stop=savetrainingplot(info, i)
    stop=false;  %prevents this function from ending trainNetwork prematurely
    if info.State=='done'   %check if all iterations have completed
    % if true
            rootfile = '../Network Progress - 4/';
            fileName = strcat(num2str(info.ValidationAccuracy),'-',num2str(i),'.jpg');

saveas(findall(groot,'Tag','NNET_CNN_TRAININGPLOT_FIGURE'),strcat(rootfile,'Trained
Plots/',fileName))  % save figure as .jpg
    end
end
```

mat2jpg.m

```
%This script is used to convert .mat files to .jpg files
clear all;

clc;

close all;

for k = 1:3064
        %Iterating over 3064 files
   load(strcat('..\Data\',num2str(k),'.mat'));

   img = cjdata.image;
   img = uint8(255 * mat2gray(img));
   label = cjdata.label;
   %Converting to .jpg files and storing in the corresponding class folder
   if label == 1
      outfile = strcat('..\Data_img\1\',num2str(k),'.jpg');
   elseif label == 2
      outfile = strcat('..\Data_img\2\',num2str(k),'.jpg');
   elseif label == 3
      outfile = strcat('..\Data_img\3\',num2str(k),'.jpg');
   end

   s = size(img);
   if ((s(1) ~= 512) || (s(2) ~= 512))
      img = imresize(img, 2);
   end

   imwrite(img, outfile);
   k    % to print the file number
end
```

Img_resize.m

```
% This script is used to resize the images which are of different sizes
clear all;
clc;
files = dir('/Users/SriKrishnaManoj/Brain-Tumour-Detection/Data_img/2/*.jpg') ;    % you
are in folder of csv files
for file = files'
    file.name
    I = imread(file.name);
    I = imresize(I, [128 128]); %Setting the files size as 128x128
    imwrite(I, file.name);
end
```