BRAIN TUMOR CLASSIFICATION

Project report submitted by-

1. RAHUL PANJA

2. RUCHIRA GHOSH

3. GOUTAMI DAS

4. ANTARIP DAS

Supervised by

Dr. Kamarujjaman

in partial fulfilment for the award of the degree of

B.Sc. in INFORMATION TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE

Year: 2020-2023

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

HARINGHATA, WEST BENGAL

BONAFIDE CERTIFICATE

Certified that this project report "**Brain Tumor Classification**" is the bonafide work of "Rahul Panja, Ruchira Ghosh, Goutami Das and Antarip Das" who carried out the project work under my supervision.

_____ _____

(Signature) (Signature)

Dr. Debasis Giri Dr. Kamarujjaman

Head of Department **SUPERVISOR**

Assistant Professor

Department of Informational Technology

Haringhata, West Bengal

_____

(Signature)

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and appreciation to all those who have contributed to the development of the brain tumor detection project.

First and foremost, we would like to thank our team members who have worked tirelessly on this project. Each team member has played a significant role in the development and implementation of the brain tumor detection system. Their dedication, expertise, and collaborative efforts have been invaluable in achieving the project's goals.

We would like to extend our deepest thanks to our project supervisor, **Dr. Kamarujjaman Sir**, for their guidance, support, and insightful feedback throughout the project. Their expertise and encouragement have been instrumental in steering the project in the right direction and ensuring its successful completion.

We would also like to acknowledge the support and resources provided by our institution or organization. The access to necessary datasets, computational infrastructure, and research facilities has greatly facilitated the development and testing of our brain tumor detection system.

Additionally, we express our gratitude to the researchers, scholars, and developers in the field of medical imaging and machine learning. Their groundbreaking work and open-source contributions have served as a foundation for our project and have inspired our approaches and methodologies.

Lastly, we would like to thank our friends and family for their understanding, encouragement, and unwavering support throughout the project. Their belief in our abilities has been a constant source of motivation during the challenging phases of development.

We recognize that this project would not have been possible without the collective efforts and support of all those mentioned above, as well as numerous others who have provided guidance, feedback, and assistance along the way. We are truly grateful for their contributions.

Thanking You,

Rahul Panja, Ruchira Ghosh, Goutami Das, Antarip Das

TABLE OF CONTENTS

CHAPTER NO.	TITLE	Page No.
	Abstract	**8**
	List of Tables	**5**
	List of Figueres	**6**
	List of Abbreviations	**7**
1	**INTRODUCTION**	**9-16**
	1.1) Overview of Brain and Brain Tumor	**10**
	1.1.1) Meningioma Tumor	**11**
	1.1.2) Giloma Tumor	**12**
	1.1.3) Pituitary Tumor	**13**
	1.2) Magnetic Resonance Image (MRI)	**14**
	1.3) Application	**15**
	1.4) Motivation	**16**
2	**EXISTING WORK AND WORKFLOW**	**17-20**
	2.1 Overview	**17-20**
	2.1.1 Working flow of ML Model	**17-18**
	2.1.2 Working flow to DL Model	**19-20**

3	**PROPOSED WORKFLOW** 3.1 Workflow 3.2 Working of Logistic Regression 3.3 Working of SVM Model 3.4 Working of Random Forest Model 3.5 Working of XG-Boost Model 3.6 Working of KNN Model 3.7 Working of CNN Model	**21-34** **21-22** **22-23** **24-25** **26-27** **28-29** **30-31** **32-34**
4	**IMPLEMENTAION AND RESULT** 4.1 Dataset Details 4.2 Tools and Technology Used 4.3 Implementation and Result	**35-44** **35** **35-37** **38-44**
5	**CONCLUSION** 5.1 Conclusion	**45** **45**
	References	**46**

LIST OF TABLES

SL. NO.	Figure	Page No.
1	Comparison of all Algorithm Used	**44**

LIST OF FIGURES

SL. NO	FIGURES	PAGE NO.
1	Basic Structure of human brain	**10**
2	Meningioma Tumor	**11**
3	Glioma Tumor	**12**
4	Pituitary Tumor	**13**
5	T1, T2 and FLAIR Image	**14**
6	Workflow	**21**
7	Working of Logistic Regression	**23**
8	Working of SVM	**25**
9	Working of Random Forest	**27**
10	Working of XG-Boost	**29**
11	Working of KNN	**31**
12	Working of CNN	**34**
13	Loading Dataset and Label Declare	**38**
14	Classified Output	**38**
15	Updated Shape	**38**
16	Split the Image Data	**38**
17	Scaling	**39**
18	Accuracy of Logistic Regression	**39**
29	Accuracy of SVM	**39**
20	Accuracy of Random Forest	**39**
21	Accuracy of XG-Boost	**39**
22	Accuracy of KNN	**40**
23	Result of Logistic Regression	**40**
24	Result of SVM	**40**
25	Result of Random Forest	**41**
26	Result of XG-Boost	**41**
27	Result of KNN	**42**
28	Train CNN Image Data	**42**
29	Accuracy of CNN	**43**
30	Result of CNN	**43**
31	Masked Image 1	**44**
32	Masked Image 2	**44**

LIST OF ABBREVIATIONS

SL. NO.	Abbreviations	Meaning
1	MRI	Magnetic Resonance Imaging
2	CNN	Convolutional Neural Network
3	SVM	Support Vector Machine
4	XG-Boost	Extreme Gradient Boosting
5	KNN	K- Nearest Neighbour
6	ML	Machine Learning
7	DL	Deep Learning
8	FLAIR	Fluid attenuated in version recovery weighted MRI
9	Algo	Algorithm
10	RNN	Recurrent Neural Network
11	GPU	Graphics Processing Units
12	TPU	Tensor Processing Units
13	ReLU	Rectified linear unit

ABSTRACT

The human brain is a vital organ, and brain tumors can lead to dysfunction. Tumors consist of excessive cells that grow uncontrollably, depriving healthy cells of essential nutrients and causing brain failure. Currently, doctors manually analyze MRI images to locate and assess brain tumors, which is time-consuming and prone to inaccuracies.

To address this issue, a computer-based brain tumor detection and classification system is proposed. It utilizes Convolutional Neural Network (CNN), Logistic regression, SVM, Random Forest, XG Boost, KNN algorithms to analyze MRI images and identify tumor blocks while classifying the tumor type. This system incorporates various image processing techniques, such as image segmentation, enhancement, and feature extraction, to aid in detecting brain tumors in cancer-affected patients. The brain tumor detection process involves four stages: Pre-Processing, Feature Extraction, and Classification. By employing image processing and neural network techniques, the system aims to enhance the accuracy and efficiency of brain tumor detection and classification in MRI images.

The results obtained from the brain tumor classification system demonstrate promising accuracy and reliability in distinguishing different types of brain tumors. The proposed system has the potential to assist radiologists and medical practitioners in making accurate and efficient diagnoses, leading to improved patient care and treatment outcomes.

Overall, this project contributes to the advancement of medical imaging analysis by developing an automated brain tumor classification system that integrates machine learning and deep learning techniques with MRI scans, paving the way for more accurate and efficient diagnosis and treatment of brain tumor patients.

1. INTRODUCTION

Brain tumor classification is a critical task in the field of medical image analysis, as it plays a vital role in the diagnosis and treatment planning for brain tumor patients. In this project, we propose a novel approach for brain tumor classification using machine learning and deep learning techniques.

The aim of this project is to develop an accurate and automated system that can classify brain tumors into different types, such as gliomas, meningiomas, and pituitary tumors, based on MRI (Magnetic Resonance Imaging) scans. The proposed system leverages the power of machine learning algorithms to analyze and extract relevant features from the MRI images, enabling effective tumor classification.

The project involves several key steps. First, a comprehensive dataset of MRI scans, including images from different tumor types, is collected and pre-processed. Various pre-processing techniques, such as reshaping, resizing etc.

Next, feature extraction is performed on the pre-processed MRI images. Different feature extraction methods, such as texture analysis and shape descriptors, are employed to capture important characteristics of the tumors. These extracted features serve as informative representations for subsequent classification.

For the classification stage, a machine learning model is trained on the extracted features. Different classifiers, such as support vector machines (SVMs), random forests, logistic regression, XG Boost, KNN, and a Deep Learning model CNN are explored and evaluated to achieve the best performance. The model is trained using a labelled dataset, where each MRI scan is associated with its corresponding tumor type.

1.1) OVERVIEW OF BRAIN AND BRAIN TUMOR

The human brain is a vital component of the nervous system, located within the skull. It serves as the command center, controlling and coordinating the functions of the entire body. Additionally, it allows humans to adapt and respond to various environmental conditions, while facilitating actions, thoughts, and emotions. Understanding the structure of the brain is crucial in comprehending its fundamental aspects.

Brain tumors are classified into two primary types: benign and malignant. Benign tumors are characterized by slow cell growth within the brain, with gliomas being a common example. Gliomas originate from non-neuronal brain cells known as astrocytes. Although less aggressive, these tumors exert significant pressure on the brain, impairing its proper function. On the other hand, malignant tumors, also referred to as secondary brain tumors, are highly aggressive and quickly spread to other tissues. They originate from cancer cells present in other parts of the body and metastasize to the brain. Examples of primary cancers leading to secondary brain tumors include lung, kidney, and bladder cancers.

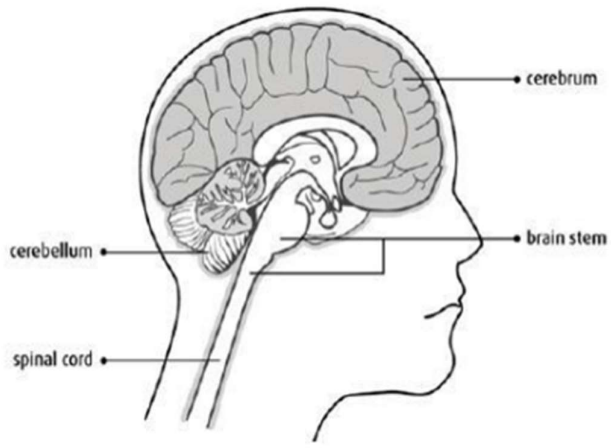

Fig.1: Basic Structure of human brain [1]

In our project we use three types of brain tumor for our classification purpose. Those are –

1.1.1) Meningioma Tumor:

Meningioma is a prevalent form of tumor that originates in the meninges, the protective membranes enveloping the brain and spinal cord. It stands as the most frequently occurring primary brain tumor, representing around 30% of cases.

These tumors typically develop from arachnoid cells, which constitute the outer layer of the meninges. Meningiomas are generally characterized by a slow growth rate and are commonly classified as benign, lacking cancerous properties.

Meningiomas can manifest at various locations along the surface of the brain and spinal cord, although they are commonly found near the convexity of the brain (outer surface) and around the parasagittal region (adjacent to large veins). The specific location of the tumor within the meninges can result in distinct symptoms and potential complications.

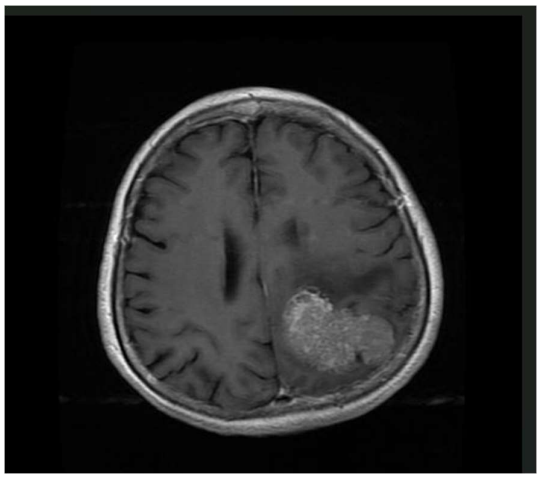

Fig.2: Meningioma Tumor

1.1.2) Glioma Tumor:

Glioma is a prevalent and primary malignant brain tumor that originates from glial cells, which are crucial for supporting and nourishing neurons in the central nervous system. Accounting for approximately 80% of malignant brain tumors, gliomas encompass several subtypes, including astrocytoma's, oligodendrogliomas, and ependymomas.

Astrocytoma's are the most common form of glioma, and they can be further classified into grades based on their aggressiveness and microscopic appearance.

Gliomas can develop in the cerebral hemispheres, which comprise the majority of the brain's volume and play a vital role in cognition, sensory processing, and motor functions. Gliomas can also affect the cerebellum, which coordinates movement and balance, or even occur within the spinal cord.

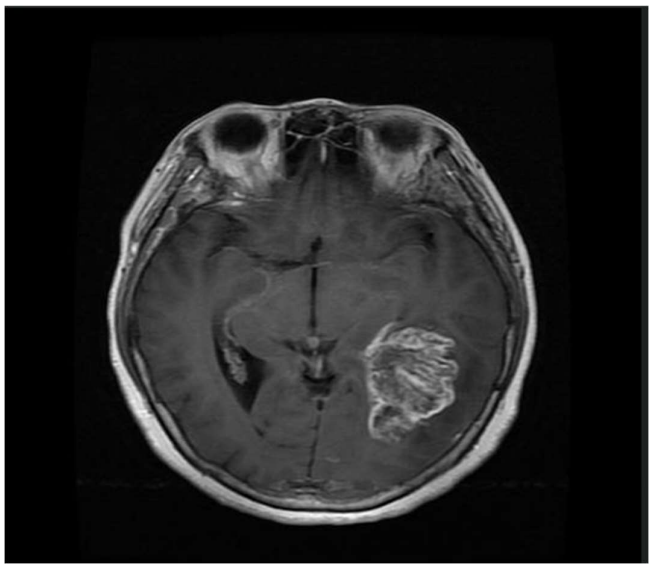

Fig.3: Glioma Tumor

1.1.3) Pituitary Tumor:

A pituitary tumor, known as a pituitary adenoma, is a growth that forms in the pituitary gland, a small gland located at the base of the brain, behind the nasal bridge.

The pituitary gland plays a crucial role in regulating various bodily functions by producing and releasing hormones that control growth, metabolism, reproduction, and the function of other endocrine glands.

Pituitary tumors are typically categorized based on their size into microadenomas (less than 10 mm in diameter) and macroadenomas (larger than 10 mm). Macroadenomas are more likely to cause symptoms due to their larger size and potential compression of surrounding structures.

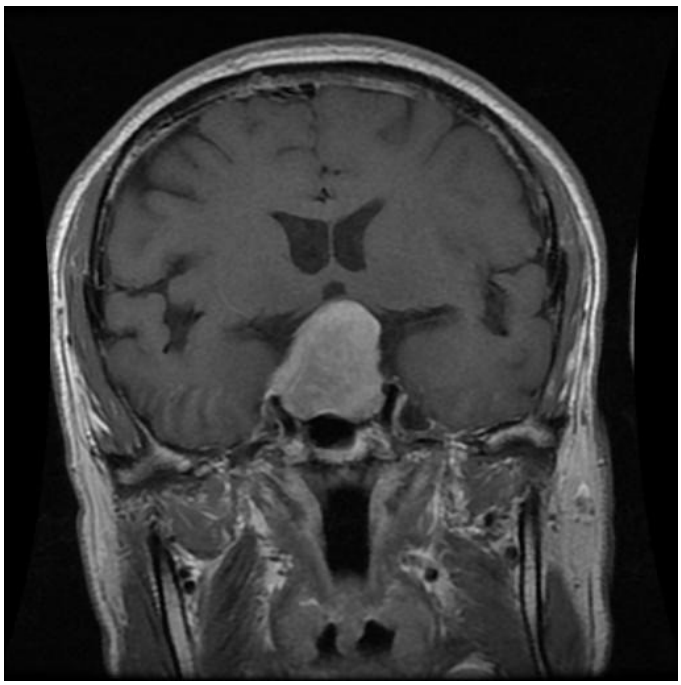

Fig. 4: Pituitary Tumor

1.2. Magnetic Resonance Imaging (MRI)

Magnetic Resonance Imaging (MRI) is a medical imaging technique that uses powerful magnets and radio waves to create detailed images of the internal structures of the body. MRI provides a non-invasive and highly accurate way to visualize different tissues and organs, including the brain.

In the context of brain imaging, an MRI scan produces high-resolution images of the brain's structure and function. It provides valuable information about the size, shape, and location of various brain regions, as well as the presence of abnormalities such as tumors, lesions, or other pathological conditions.

MRI images of the brain are captured in multiple planes (slices) and can be reconstructed in three dimensions to provide a comprehensive view of the brain's anatomy. Different types of MRI sequences, such as T1-weighted, T2-weighted, and contrast-enhanced scans, are used to visualize different aspects of brain tissue and pathology.

Overall, MRI imaging plays a crucial role in the diagnosis, treatment planning, and monitoring of brain diseases by providing detailed and accurate information about the structure and function of the brain.

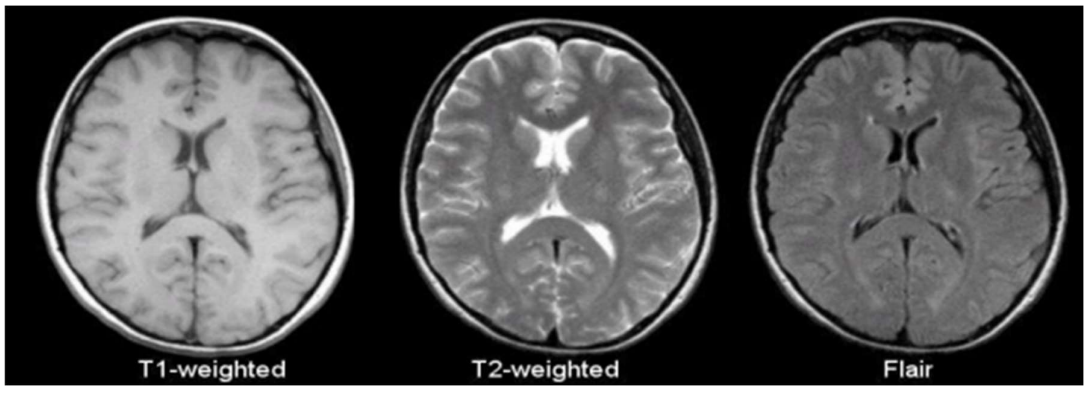

Fig.5: T1, T2 and Flair image [2]

1.3 Application:

The application of a brain tumor classification project holds significant potential in the field of healthcare and medical imaging, offering a range of practical uses. Some key applications include:

Early Detection: By analyzing medical imaging data, the classification model can aid in the early detection of brain tumors. Early diagnosis enables prompt intervention and treatment, leading to improved patient outcomes.

Treatment Planning: Accurate classification of brain tumors assists in determining the most suitable treatment plan for individual patients. Healthcare professionals can customize treatment strategies, such as surgery, radiation therapy, chemotherapy, or a combination thereof, based on the specific tumor type and characteristics.

Surgical Guidance: Brain tumor classification provides valuable information for neurosurgeons during surgical procedures. It helps identify tumor boundaries and determine the extent of surgical resection.

Research and Clinical Trials: Brain tumor classification projects contribute to ongoing research in neuro-oncology. These models assist in analyzing large datasets, identifying patterns, and uncovering new insights into tumor behavior, treatment responses, and potential prognostic factors. Additionally, they enable patient stratification for clinical trials, enhancing the selection and evaluation of appropriate treatment approaches.

In summary, brain tumor classification projects have the potential to enhance the accuracy and efficiency of brain tumor diagnosis, treatment planning, surgical interventions, and progress monitoring. They support healthcare professionals in delivering personalized and effective care to individuals with brain tumors, ultimately improving patient management and quality of life.

1.4 Motivation:

The primary motivation behind brain tumor detection is not only to identify the presence of a tumor but also to classify its type accurately. This capability proves invaluable in situations where it is necessary to determine whether a tumor is positive or negative. The proposed project aims to develop a system that utilizes computer-based procedures to detect tumor blocks and classify the type of tumor using various machine learning (ML) and deep learning (DL) models.

By leveraging ML models such as logistic regression, support vector machines (SVM), random forest, as well as DL models like convolutional neural networks (CNN), the system can effectively analyze MRI images of different patients. Through a combination of image processing techniques and pattern recognition algorithms, relevant features are extracted from the images. These features are then utilized by the ML and DL models to detect the presence of a tumor and classify its specific type, such as meningioma, glioma, or pituitary tumor.

The significance of this project lies in its potential to assist in both tumor detection and classification. By accurately determining whether a tumor is positive or negative, it enables timely medical intervention and treatment planning. Additionally, the ability to classify the tumor type facilitates personalized treatment approaches, allowing healthcare professionals to tailor therapies based on specific tumor characteristics.

By employing ML and DL models and leveraging the power of neural networks, this project contributes to the field of medical image analysis. It provides healthcare professionals with a reliable tool for accurate brain tumor detection and classification, enhancing the overall efficiency and effectiveness of diagnosis and treatment.

2. EXISTING WORK AND PROPOSED WORKFLOW

2.1) Overview:

2.1.1) Working flow of ML model:

- **Data Collection:** Acquire a dataset containing relevant data for the ML task at hand. This dataset should consist of input variables (features) and corresponding output variables (labels) that the model will learn from.

- **Data Preprocessing:** Clean and preprocess the data to ensure its quality and suitability for training. This step may involve handling missing values, outlier detection and removal, feature scaling or normalization, and categorical variable encoding.

- **Dataset Splitting:** Divide the dataset into distinct subsets, typically a training set, a validation set (optional), and a test set.

- **Feature Engineering**: If necessary, perform feature engineering techniques to enhance the model's performance. This may include feature selection, dimensionality reduction, or generating new features based on domain knowledge.

- **Model Selection:** Select a suitable ML model that aligns with the task requirements, such as

linear regression, decision trees, support vector machines (SVM), random forests, or neural networks.

- **Model Training:** Train the chosen ML model using the training dataset. The model learns patterns and relationships between the input features and output labels during this phase. The specific training algorithm and optimization technique depend on the selected model.

- **Model Testing:** Assess the model's performance on the test set, providing an unbiased estimate of its generalization capability. This step evaluates how well the model performs on unseen data and offers insights into its real-world applicability.

- **Model Deployment:** Once satisfied with the model's performance, deploy it in a production environment to make predictions on new, unseen data. Continuously monitor the model's performance over time and consider retraining or updating it periodically as new data becomes available.

It's crucial to emphasize that these steps serve as general guidelines, and the specific details may vary depending on the ML task, dataset characteristics, and chosen algorithms.

2.1.2) Working flow of DL model:

- **Data Collection:** Gather a dataset specific to your deep learning task, including the input data (e.g., images, text, time series) and corresponding target labels. Ensure the dataset is diverse, representative, and of sufficient size for effective training.

- **Data Preprocessing:** Clean and preprocess the data to ensure its quality and suitability for training. This step involves tasks such as removing noise, handling missing values, normalizing or standardizing features, and performing data augmentation techniques to increase the dataset's diversity.

- **Dataset Splitting**: Divide the dataset into training, validation, and testing sets. Allocate the majority of the data for training, a smaller portion for validation to tune the model's hyperparameters, and a separate portion for unbiased evaluation during testing.

- **Model Architecture Design:** Choose an appropriate deep learning architecture based on the nature of your data and the specific task. Consider architectures such as convolutional neural networks (CNNs) for image data, recurrent neural networks (RNNs) for sequential data, or transformer networks for natural language processing tasks.

- **Model Compilation:** Configure the model by selecting a suitable loss function, optimizer, and evaluation metrics. The loss function measures the discrepancy between predicted and target values, the optimizer determines how the model's parameters are

updated during training, and the evaluation metrics provide insights into the model's performance.

- **Model Training:** Train the deep learning model using the training dataset. Feed the input data into the model, compute the loss, and update the model's parameters through backpropagation. Training typically involves iterating over the dataset in mini-batches for multiple epochs.

- **Model Testing:** Evaluate the final trained model on the testing set to measure its performance on unseen data. This step provides an unbiased estimate of the model's ability to generalize and helps assess its real-world applicability.

- **Model Deployment:** Once satisfied with the model's performance, deploy it in a production environment to make predictions on new, unseen data. Set up the necessary infrastructure, such as servers or cloud platforms, to serve the model and monitor its performance for potential updates or retraining.

Remember, these steps provide a general framework, and specific details may vary depending on the deep learning task, dataset, and chosen model architecture. It's crucial to experiment, iterate, and continuously improve the model to achieve the best results.

3.Proposed Workflow:

3.1) Workflow:

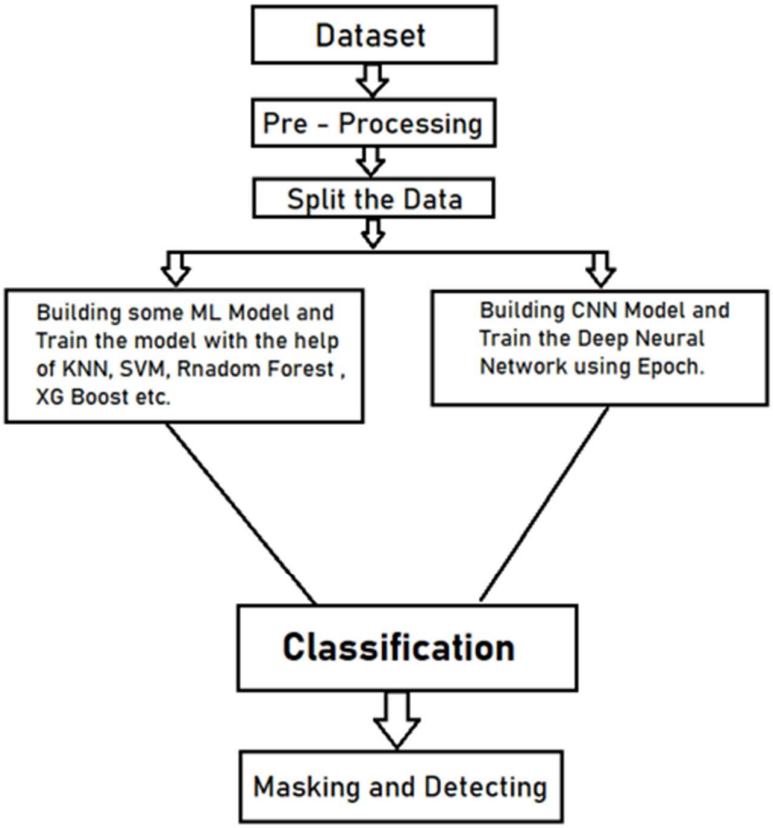

Fig.6: Workflow

The proposed system has mainly six modules. Datasets, Pre-processing, split the data, build ML models using some ML algorithms as well as build CNN model train Deep Neural Network for epochs, and classification. In dataset we can take Multiple MRI images and take one as input image. In pre-Processing image to encoded the label and resize the image. Then train the data using all the ML and DL models.

Build CNN model train deep neural network for epochs. Then classified the image as different kind of tumors. If the train model successfully classified the image, then the outcomes will be accurate. After classify the image we do masking to segmented the tumor from the brain image.

3.2) Working of Logistics Regression model:

Below the usage of logistic regression for classification using scikit-learn. Let's break it down step by step:

- **Importing Required Libraries:** The code begins by importing essential libraries such as LogisticRegression, SVC, RandomForestClassifier, make_classification, and warnings. These libraries provide the necessary tools for implementing and evaluating the logistic regression model.

- **Data Preparation:** Prior to fitting the logistic regression model, the code assumes that the training and testing datasets (xtrain, ytrain, xtest, ytest) have been prepared. The feature data is stored in xtrain and xtest, while the corresponding labels are stored in ytrain and ytest.

- **Ignoring Warnings:** The warnings.filterwarnings('ignore') statement is included to suppress any warning messages that may arise during the code execution.

- **Logistic Regression Model Initialization**: The logistic regression model is initialized using the LogisticRegression class from scikit-learn. In this case, the regularization parameter C is set to 0.01. A smaller C value indicates stronger regularization, which helps prevent overfitting.

- **Model Training:** The logistic regression model is trained using the fit() method by passing the training data (xtrain, ytrain) as arguments. This process involves finding the

optimal coefficients or weights that minimize the logistic loss function.

- **Model Evaluation:** The code prints two metrics to evaluate the trained logistic regression model: the training score and the testing score. The score() method is utilized to calculate the accuracy of the model on the respective datasets.

    - The training score is obtained by passing the training data (xtrain, ytrain) to the score() method of the logistic regression model.
    - The testing score is obtained by passing the testing data (xtest, ytest) to the score() method.

In summary, the code demonstrates a basic implementation of logistic regression for binary classification using scikit-learn. It trains the model on the provided datasets and evaluates its performance using accuracy scores. Logistic regression aims to find the optimal coefficients to make predictions on unseen data.

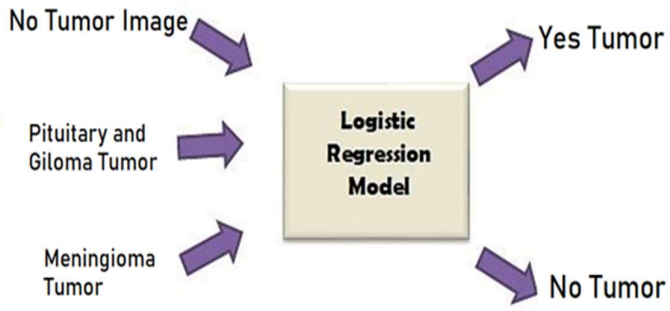

Fig.7: Working of Logistic Regression [3]

3.3) Working of SVM model:

Below the implementation and evaluation of a Support Vector Machine (SVM) model using the scikit-learn library. Let's explore how SVM works:

- **Importing Required Libraries:** The code begins by importing the necessary libraries, including SVC from sklearn.svm, make_classification from sklearn.datasets, and warnings to suppress any warning messages during execution.

- **Data Preparation:** Before fitting the SVM model, it is assumed that the training and testing datasets (xtrain, ytrain, xtest, ytest) have been prepared.

- **SVM Model Initialization:** The SVM model is initialized using the SVC class from scikit-learn. The parameters kernel, C, and gamma are specified as follows:

    - The 'rbf' kernel refers to the Radial Basis Function (RBF) kernel, which is commonly used for solving non-linear classification problems.
    - The C parameter controls the regularization strength of the SVM model, where a higher value indicates less regularization.
    - The gamma parameter determines the kernel coefficient, with 'scale' indicating that it will be calculated based on the inverse of the number of features.

- **Model Training:** The SVM model is trained by calling the fit() method and passing the training data (xtrain, ytrain) as arguments. During training, the SVM model optimizes its hyperparameters to find the optimal decision boundary that separates the classes in the training data.

- **Model Evaluation:** The code prints two metrics to evaluate the trained SVM model: the training score and the testing score. The score() method calculates the accuracy of the model on the respective datasets.

- The training score is obtained by passing the training data (xtrain, ytrain) to the score() method of the SVM model.
- The testing score is obtained by passing the testing data (xtest, ytest) to the score() method.

- Both scores range from 0 to 1, with higher values indicating better performance. These scores provide an indication of how accurately the SVM model predicts the labels for the training and testing data.

In summary, the provided code demonstrates the implementation of SVM for classification using scikit-learn. It initializes the SVM model, trains it on the training data, and evaluates its performance using accuracy scores on the training and testing datasets.

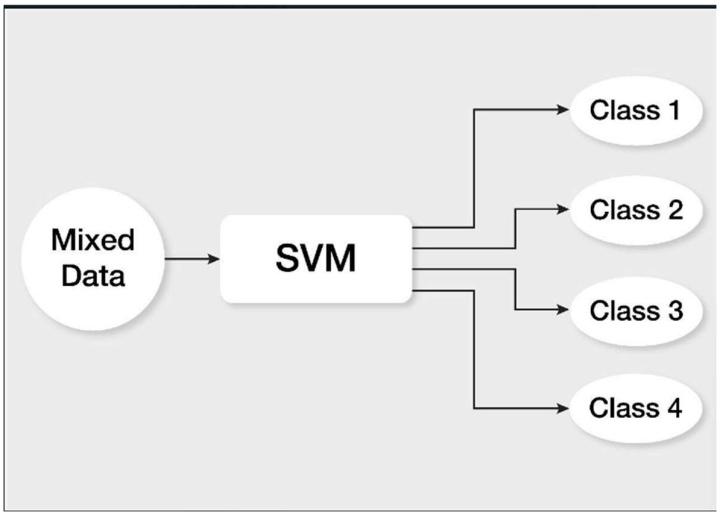

Fig.8: Working of SVM [4]

3.4) Working of Random Forest model:

Below the code snippet demonstrates the implementation and evaluation of a Random Forest Classifier using scikit-learn. Random Forest is an ensemble learning algorithm that combines the predictions of multiple decision trees to make a final prediction. Here's how it works:

- **Random Forest Classifier Initialization**: The Random Forest Classifier is initialized with the RandomForestClassifier class from scikit-learn. In this code, the number of estimators (n_estimators) is set to 500, which specifies the number of decision trees to be included in the random forest ensemble.

- **Model Training:** The random forest model is trained using the fit() method by passing the training data (xtrain, ytrain) as arguments. During training, the random forest builds an ensemble of decision trees. Each decision tree is trained on a randomly selected subset of the training data, and a random subset of features is considered for each split. This randomness helps to reduce overfitting and increase the model's ability to generalize to unseen data.

- **Ensemble Learning:** Random Forest combines the predictions of all the decision trees in the ensemble to make the final prediction. In a classification task, the random forest uses a majority voting scheme. Each decision tree independently predicts the class label, and the final prediction is determined by the majority vote among all the trees.

- **Model Evaluation:** After training, the performance of the random forest model is evaluated using the score() method. The score() method calculates the accuracy

of the model on both the training data (xtrain, ytrain) and the testing data (xtest, ytest).

- The training score is obtained by passing the training data (xtrain, ytrain) to the score() method of the random forest model.
- The testing score is obtained by passing the testing data (xtest, ytest) to the score() method.
- The scores range from 0 to 1, with higher values indicating better performance. The scores reflect how accurately the random forest model predicts the labels for the training and testing data.

In summary, the provided code initializes and trains a Random Forest Classifier with 500 decision trees. The model is evaluated on both the training and testing datasets, providing insights into its performance and generalization ability.

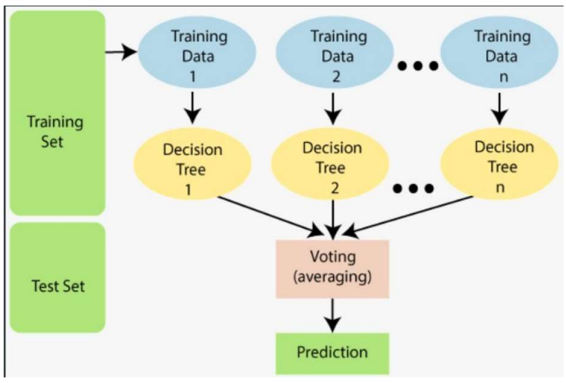

Fig.9: Working of Random Forest [5]

3.5) Working of XGboost model:

Below the implementation and evaluation of the XGBoost (eXtreme Gradient Boosting) model using the xgboost library. Let's break down the code and understand how XGBoost works:

- **Importing Required Libraries:** The code begins by importing the necessary libraries, including xgb from xgboost.

- **XGBoost Classifier Initialization:** The XGBoost classifier is initialized using the XGBClassifier class from xgboost. This classifier comes with default hyperparameters that are optimized for general cases. However, you can also customize the hyperparameters according to your specific requirements.

- **Model Training:** The XGBoost model is trained using the fit() method by providing the training data (xtrain, ytrain) as arguments. During training, XGBoost constructs an ensemble of weak prediction models, typically decision trees, in a sequential manner.

- **Gradient Boosting:** XGBoost is an implementation of the gradient boosting algorithm. It creates a sequence of decision trees, where each subsequent tree corrects the mistakes made by the previous trees. The trees are added one at a time, and the model learns from the errors (residuals) of the previous trees to make better predictions.

- **Model Evaluation:** After training, the performance of the XGBoost model is evaluated using the score() method. This method calculates the accuracy of the model on both the training data (xtrain, ytrain) and the testing data (xtest, ytest).

    - The training score is obtained by passing the training data (xtrain, ytrain) to the score() method of the XGBoost model.
    - The testing score is obtained by passing the testing data (xtest, ytest) to the score() method.
    - Both scores range from 0 to 1, with higher values indicating better performance. These scores provide an assessment of how accurately the XGBoost model predicts the labels for the training and testing data.

In summary, the provided code initializes and trains an XGBoost classifier using the xgboost library. The model's performance is evaluated on both the training and testing datasets, providing insights into its accuracy and generalization capabilities.

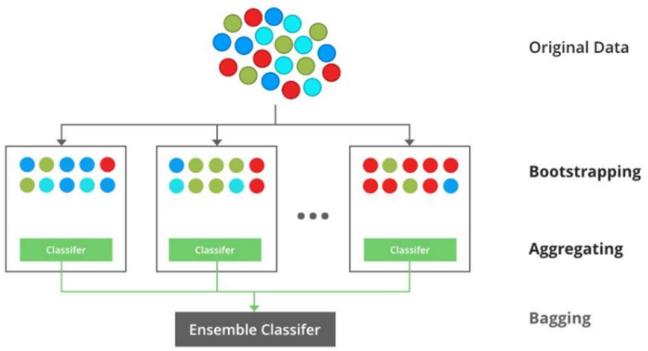

Fig.10: Working of XG-Boost [6]

3.6) Working of KNN model:

Below the implementation and evaluation of the K-Nearest Neighbors (KNN) model using the scikit-learn library. Here's a breakdown of how the KNN model works:

- **Importing Required Libraries:** The code starts by importing the necessary library, specifically the KNeighborsClassifier class from sklearn.neighbors.

- **KNN Classifier Initialization:** The KNN classifier is initialized using the KNeighborsClassifier class from scikit-learn. In this case, the parameter n_neighbors is set to 5, indicating that the model will consider the labels of the five nearest neighbors in the training data to make predictions.

- **Model Training:** The KNN model is trained using the fit() method by passing the training data (xtrain, ytrain) as arguments. During the training phase, the KNN model builds a lookup table of the training data to quickly find the K nearest neighbors for any given test data point.

- **Nearest Neighbor Classification:** In KNN, classification is performed based on the majority vote of the K nearest neighbors. When making predictions for a test data point, the KNN model identifies the K nearest neighbors in the training data and assigns the class label that is most common among these neighbors. The choice of K determines the level of model complexity and can be tuned based on the specific problem.

- **Model Evaluation**: After training, the performance of the KNN model is evaluated using the score() method. The score() method calculates the accuracy of the model on both the training data (xtrain, ytrain) and the testing data (xtest, ytest).

    - The training score is obtained by passing the training data (xtrain, ytrain) to the score() method of the KNN model.
    - The testing score is obtained by passing the testing data (xtest, ytest) to the score() method.
    - Both scores range from 0 to 1, with higher values indicating better performance. These scores reflect how accurately the KNN model predicts the labels for the training and testing data.

In summary, the provided code initializes and trains a KNN classifier with 5 neighbors using scikit-learn. The model's performance is then evaluated on both the training and testing datasets, providing insights into its accuracy and generalization ability.

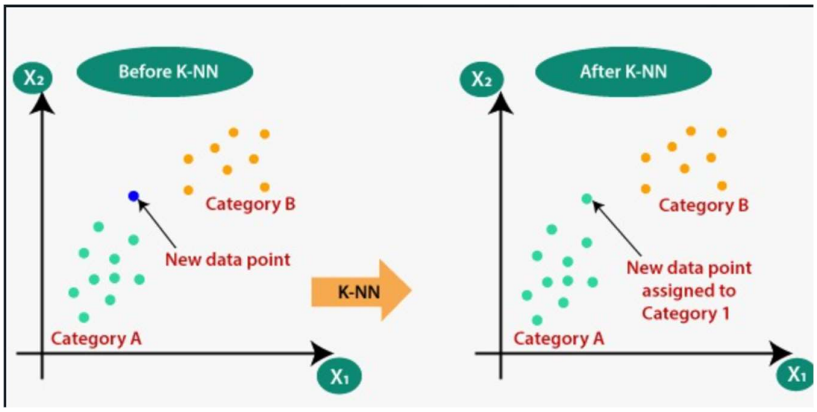

Fig.11: Working of KNN [7]

3.7) Working of CNN Model:

Below the implementation of a Convolutional Neural Network (CNN) using the TensorFlow and Keras libraries. Let's break down the code and explain how the CNN model works:

- **Importing Required Libraries:** The code begins by importing the necessary libraries, including TensorFlow and the layers module from TensorFlow's Keras API.

- **Input Shape Definition:** The input_shape variable is defined to represent the shape of the input images. In this case, it is set to (300, 300, 1), assuming grayscale images with a size of 300x300 pixels.

- **Loading and Preprocessing Images:** The code creates empty lists to store the images and labels. It loops over the classes and loads the images from each class, resizes them to the desired input_shape, and adds them to the X list. The corresponding labels are added to the Y list. The images are also reshaped to include the channel dimension (1 for grayscale images).

- **Data Normalization:** The X array is converted to a NumPy array, and the pixel values are normalized by dividing them by 255.0. This step scales the pixel values between 0 and 1, which helps the model learn more effectively.

- **Splitting the Data:** The data is split into training and testing sets using the train_test_split function from scikit-learn. The X and Y arrays are split into x_train, x_test, y_train, and y_test, which will be used for training and evaluating the model, respectively.

- **CNN Model Architecture:** The code defines the architecture of the CNN model using the Sequential API from Keras. The model consists of several layers:
  - **Convolutional layers:** The model starts with a Conv2D layer that applies 32 filters of size (3, 3) and uses the ReLU activation function. This layer takes the input_shape as the input layer. Another Conv2D layer follows with 64 filters of size (3, 3) and ReLU activation.
  - **MaxPooling layers:** Two MaxPooling2D layers with pool size (2, 2) are added to downsample the feature maps and capture the most important information.
  - **Dropout layers:** Two Dropout layers are inserted after the MaxPooling layers to reduce overfitting. They randomly set 25% of the neurons to 0 during training, helping the model generalize better to unseen data.
  - **Flatten layer:** This layer flattens the 2D feature maps into a 1D vector, preparing the data for the fully connected layers.
  - **Fully connected layers:** The flattened features are connected to a Dense layer with 128 units and the ReLU activation function. A Dropout layer with a dropout rate of 0.5 follows, preventing overfitting. Finally, a Dense layer with the number of units equal to the number of classes and a SoftMax activation function is added to produce the final class probabilities.
- **Model Compilation:** The model is compiled by specifying the optimizer, loss function, and metrics to be used during training. In this case, the Adam optimizer,

sparse_categorical_crossentropy loss function, and accuracy metric are chosen. The optimizer updates the model's weights based on the calculated gradients, the loss function measures the model's performance, and the accuracy metric evaluates the model's accuracy during training.

- **Model Training:** The model is trained using the fit() function. The x_train and y_train datasets are used for training, and the x_test and y_test datasets are used for validation during the training process. The training is performed for 50 epochs with a batch size of 50, meaning the model will iterate over the training data 50 times in batches of size 50.

In summary, the provided code implements a CNN model for image classification. It loads and preprocesses the images, defines the model architecture, compiles the model with an optimizer, loss function, and metrics, and trains it on the provided dataset. The trained model can then be used to make predictions on new images.

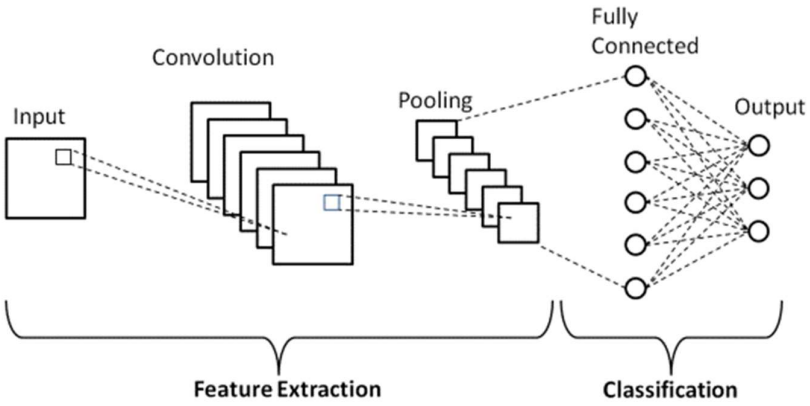

Fig.12: Working of CNN [8]

4. DATASET, IMPLEMENTATION AND RESULT

4.1) DATASET DETAIL

The dataset has 3200 images with different types of tumors like glioma, pituitary and meningioma and also including images which has no Tumor.

- **Brain Tumor Classification (MRI) –** https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri [9]

4.2) TOOLS & TECHNOLOGY USED

➢ Python:

Python was chosen as the language for this project due to several reasons. Firstly, Python has a large and active community, making it easy to find solutions and answers to any encountered issues through resources like Stack Overflow. Being one of the most popular languages on the platform ensures that there is a wealth of knowledge available.

Secondly, Python offers a wide range of powerful tools specifically designed for scientific computing. Packages such as NumPy, Pandas, and SciPy provide extensive functionality and are well-documented. These packages significantly reduce the amount of code needed for various scientific computations, leading to faster iterations and development.

Additionally, the lack of clear return type documentation in standard Python libraries can introduce trial-and-error testing during the learning process, making it more challenging to work with new packages or libraries.

➢Google Collab:

Google Collab, short for Google Collaboratory, is a cloud-based Jupyter notebook environment provided by Google. It allows users to write and execute Python code directly in their web browsers without requiring any setup or installation.

Here are some key features and benefits of Google Collab:

- **Free and Cloud-Based:** Google Collab is available for free and runs entirely on Google's cloud infrastructure. This eliminates the need for users to install and configure Python or any libraries on their local machines.

- **Hardware Acceleration**: Collab provides access to GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units) for running code that requires high computational power, such as deep learning tasks. This allows users to train models faster and work with larger datasets efficiently.

- **Pre-installed Libraries:** Collab comes with many popular Python libraries pre-installed, including TensorFlow, PyTorch, and scikit-learn. This makes it convenient for users to start working on data analysis, machine learning, and other projects without worrying about library installations.

- **Easy Data Access and Integration**: Collab provides seamless integration with Google Drive, allowing users to easily access and load datasets or save their work. It also supports importing data from other sources and provides options for data manipulation and visualization.

- **Collaboration and Sharing:** Collab allows multiple users to collaborate on the same notebook in real-time. It supports simultaneous editing, comments, and discussions, making it a useful tool for team projects. Notebooks can also be shared with others via a simple link.

Overall, Google Collab provides an accessible and powerful environment for Python programming and data analysis. Its cloud-based nature, integration with popular libraries, and collaboration features make it a preferred choice for researchers, students, and professionals working on machine learning, data analysis, and other computational tasks.

➢ Libraries and Packages –

- Numpy
- Pandas
- CV2
- MatplotLib
  - Imshow
  - imread
- sklearn.model_selection
- sklearn.metrics
- sklearn.linear_model(LogisticRegression)
- sklearn.svm(SVC)
- sklearn.ensemble(RandomForestClassifier)
- sklearn.datasets (make_classification)
- xgboost
- sklearn.neighbors (KNeighborsClassifier)
- Tensorflow
- Image

4.3) Implementation and Result:

- **Load the data and classify the label**

```
path = os.listdir('/content/drive/MyDrive/Brain Tumor Dataset')
classes = {'no_tumor': 0, 'pituitary_tumor': 1, 'meningioma_tumor': 2, 'glioma_tumor': 3}
```

Fig.13: Loading Dataset and Label classification

- **Classified output**

```
np.unique(Y)
```

```
array([0, 1, 2, 3])
```

Fig.14: Classified Output

- **Updated Shape**

```
X.shape, X_updated.shape
```

```
((2827, 300, 300), (2827, 90000))
```

Fig.15: Updated Shape

- **Split the Image Data**

```
xtrain.shape, xtest.shape
```

```
((2120, 90000), (707, 90000))
```

Fig.16: Split the Data

- **Scaling**

```
print(xtrain.max(), xtrain.min())
print(xtest.max(), xtest.min())
xtrain = xtrain/255
xtest = xtest/255
print(xtrain.max(), xtrain.min())
print(xtest.max(), xtest.min())
```

```
255 0
255 0
1.0 0.0
1.0 0.0
```

Fig.17: Scaling

- **Logistics Regression Score:**

```
Training Score: 0.9915094339622641
Testing Score: 0.751060820367751
```

Fig.18: Accuracy of Logistic Regression

- **SVM Score:**

```
Training Score: 0.9372641509433962
Testing Score: 0.8048090523338048
```

Fig.19: Accuracy of SVM

- **Random Forest Score:**

```
Training Score: 1.0
Testing Score: 0.8868458274398868
```

Fig.20: Accuracy of Random Forest

- **XG-Boost Score:**

```
Training Score: 1.0
Testing Score: 0.8769448373408769
```

Fig.21: Accuracy of XG-Boost

- **KNN Score:**

```
Training Score: 0.8759433962264151
Testing Score: 0.7553041018387553
```

Fig.22: Accuracy of KNN

- **Logistics Regression test result:**

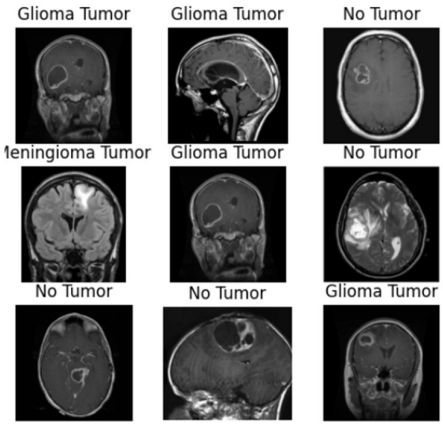

Fig.23: Result of Logistic Regression

- **SVM test result:**

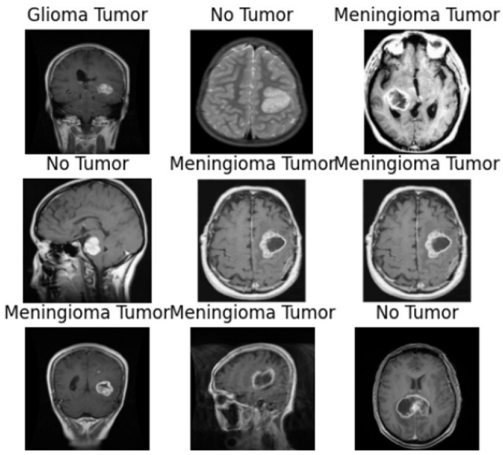

Fig.24: Result of SVM

- **Random Forest result:**

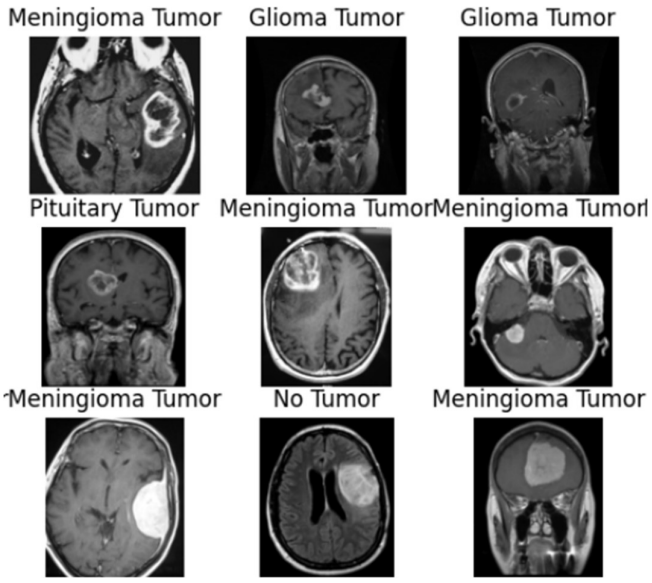

Fig.25: Result of Random Forest

- **XG-Boost:**

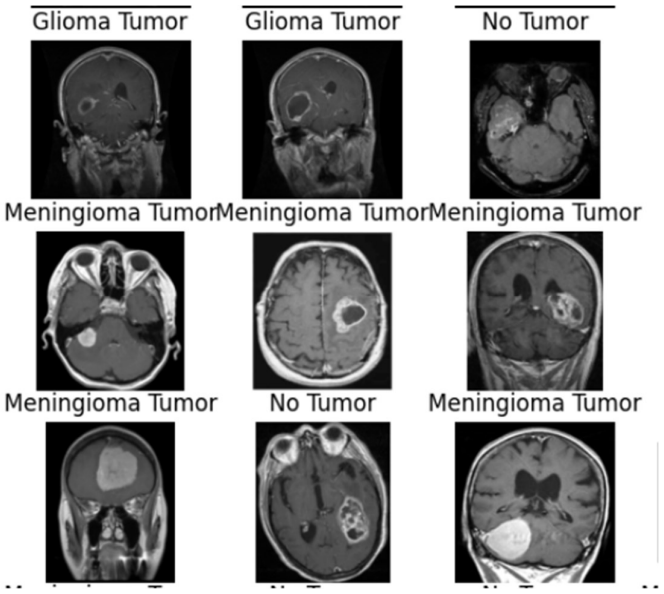

Fig.26: Result of XG-Boost

- **KNN:**

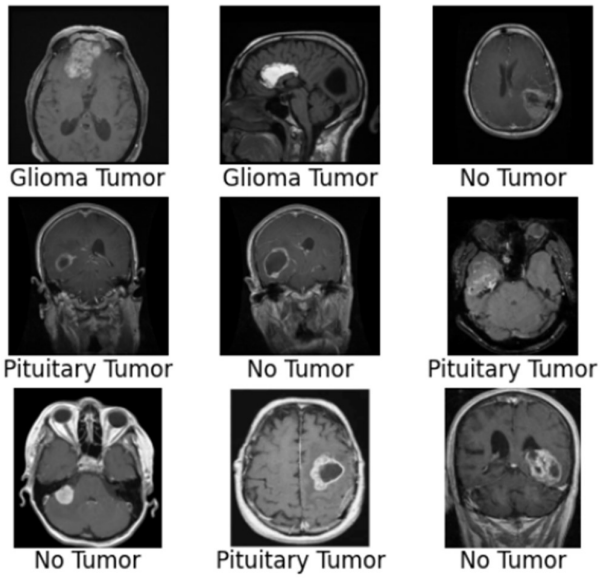

Fig.27: Result of SVM

- **Train Data for CNN**

```
43/43 [==============================] - 9s 207ms/step - loss: 0.0172 - accuracy: 0.9934 - val_loss: 0.7208 - val_accuracy: 0.8543
Epoch 33/50
43/43 [==============================] - 9s 200ms/step - loss: 0.0481 - accuracy: 0.9840 - val_loss: 0.7465 - val_accuracy: 0.8388
Epoch 34/50
43/43 [==============================] - 8s 198ms/step - loss: 0.0269 - accuracy: 0.9901 - val_loss: 0.7349 - val_accuracy: 0.8444
Epoch 35/50
43/43 [==============================] - 9s 206ms/step - loss: 0.0298 - accuracy: 0.9915 - val_loss: 0.8712 - val_accuracy: 0.8359
Epoch 36/50
43/43 [==============================] - 9s 199ms/step - loss: 0.0180 - accuracy: 0.9958 - val_loss: 0.7596 - val_accuracy: 0.8571
Epoch 37/50
43/43 [==============================] - 9s 200ms/step - loss: 0.0146 - accuracy: 0.9929 - val_loss: 0.8552 - val_accuracy: 0.8515
Epoch 38/50
43/43 [==============================] - 9s 203ms/step - loss: 0.0169 - accuracy: 0.9943 - val_loss: 0.8536 - val_accuracy: 0.8628
Epoch 39/50
43/43 [==============================] - 8s 197ms/step - loss: 0.0332 - accuracy: 0.9896 - val_loss: 0.8892 - val_accuracy: 0.8274
Epoch 40/50
43/43 [==============================] - 9s 200ms/step - loss: 0.0156 - accuracy: 0.9948 - val_loss: 0.9246 - val_accuracy: 0.8289
Epoch 41/50
43/43 [==============================] - 9s 205ms/step - loss: 0.0118 - accuracy: 0.9962 - val_loss: 0.9057 - val_accuracy: 0.8628
Epoch 42/50
43/43 [==============================] - 9s 201ms/step - loss: 0.0146 - accuracy: 0.9948 - val_loss: 0.7745 - val_accuracy: 0.8699
Epoch 43/50
43/43 [==============================] - 9s 201ms/step - loss: 0.0192 - accuracy: 0.9948 - val_loss: 0.8734 - val_accuracy: 0.8458
Epoch 44/50
43/43 [==============================] - 9s 205ms/step - loss: 0.0215 - accuracy: 0.9929 - val_loss: 0.7916 - val_accuracy: 0.8501
Epoch 45/50
43/43 [==============================] - 9s 206ms/step - loss: 0.0183 - accuracy: 0.9958 - val_loss: 0.8878 - val_accuracy: 0.8515
Epoch 46/50
43/43 [==============================] - 9s 207ms/step - loss: 0.0123 - accuracy: 0.9962 - val_loss: 0.8969 - val_accuracy: 0.8458
Epoch 47/50
43/43 [==============================] - 9s 207ms/step - loss: 0.0142 - accuracy: 0.9939 - val_loss: 1.0307 - val_accuracy: 0.8430
Epoch 48/50
43/43 [==============================] - 9s 201ms/step - loss: 0.0203 - accuracy: 0.9939 - val_loss: 0.7810 - val_accuracy: 0.8543
Epoch 49/50
43/43 [==============================] - 9s 203ms/step - loss: 0.0420 - accuracy: 0.9868 - val_loss: 0.7243 - val_accuracy: 0.8388
Epoch 50/50
43/43 [==============================] - 9s 200ms/step - loss: 0.0265 - accuracy: 0.9901 - val_loss: 0.7313 - val_accuracy: 0.8614
<keras.callbacks.History at 0x7f7b3ec16b00>
```

Fig.28: Train CNN image data

- **CNN Score:**

```
_, test_accuracy = model.evaluate(x_test, y_test)
print('Test Accuracy:', test_accuracy)
```

```
23/23 [==============================] - 1s 32ms/step - loss: 0.7313 - accuracy: 0.8614
Test Accuracy: 0.8613861203193665
```

Fig.29: Accuracy of CNN

- **CNN Test Result:**

Fig.30: Result of CNN

- **Masked and Segmented Image:**

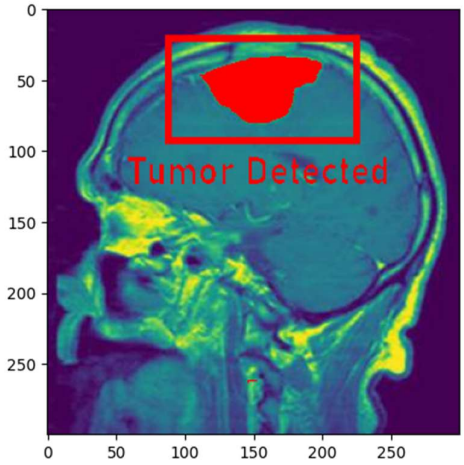

Fig.31: Masked Image 1

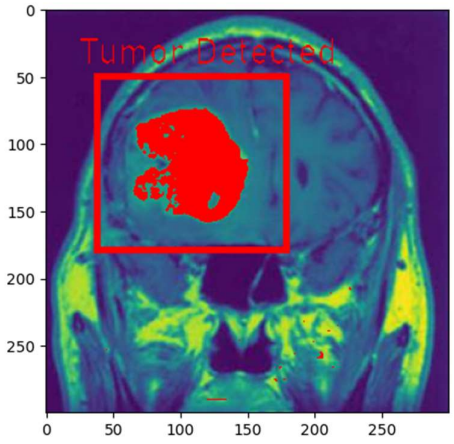

Fig.32: Masked Image 2

- **Comparison:**

Algorithms	Accuracy
Random Forest	**88.68%**
XG-Boost	**87.69%**
CNN (With 50 Epoch)	**86.13%**
SVM	**80.48%**
KNN	**75.53%**
Logistic Regression	**75.1%**

Table.1: Comparison of all Algorithm Used

5. CONCLUSION

5.1) Conclusion:

For this project, we have studied about image processing techniques like pre-processing, image segmentation. We also studied Machine Learning and Deep learning algorithm techniques. For Machine Learning algorithm we used Random Forest, SVM, XG-Boost, KNN, Logistic Regression. In Deep Learning we used CNN model which is one the most fundamental algorithms for Brain Tumor classification.

In this system we have detect the tumor is present or not if the tumour is present then we further classified them which type of tumor is that, this is the main motive of out project. Then we compare all the algorithms and got the best accuracy on Random Forest followed by XG-Boost and CNN.

However, not every task is said to be perfect in this development field even more improvement may be possible in this application. Throughout this project we have learned a lot of techniques and compare them based on the classification results.

References

[1] Obtainable Online: www.cancer.ca/~/media/CCE 10/08/2015.

[2] Preston, D. c. (2006). Magnetic Resonance Imaging (MRI) of the Brain and Spine from Basics. casemed.case.edu.

[3] https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

[4] https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

[5] https://www.javatpoint.com/machine-learning-random-forest-algorithm

[6] https://www.geeksforgeeks.org/xgboost/

[7] https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

[8] https://www.upgrad.com/blog/basic-cnn-architecture/

[9] https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri