IDENTIFICATION OF ALZHEIMER'S DISEASE FROM MRI IMAGES USING MACHINE LEARNING TOOLS

A PROJECT REPORT

Submitted By

Anubhab Halder, Roll No- 12621018010, Reg No-211260130810001
Soumya Chowdhury, Roll No- 12622018073, Reg No- 221260121009
Kingshuk Sarkar, Roll No- 12621018020, Reg No- 211260130810015
Anshuman Prabhakar, Roll No- 12621018009, Reg No- 211260130810031

Under the supervision of

Prof. (Dr.) Sujay Saha
Dept. of Computer Science and Engineering
(Artificial Intelligence and Machine Learning)
Heritage Institute of Technology, Kolkata

in partial fulfilment for the award of the degree
of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING
(Artificial Intelligence & Machine Learning)

HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA

An autonomous Institute under
Maulana Abdul Kalam Azad University of Technology
Formerly Known as a
West Bengal University of Technology

May, 2025

ACKNOWLEDGEMENT

We would take this opportunity to thank Prof. (Dr.) Basab Chaudhuri, Principal Heritage Institute of Technology for providing us with all the necessary facilities to make our project work successful.

We would like to thank our Head of the Department and project coordinator Prof. (Dr.) Sujay Saha for constantly supporting and guiding us for giving us invaluable insights. His guidance and his words of encouragement motivated us to achieve our goal and impetus to excel.

We would also like to express our sincere gratitude to Assistant Prof. Saikat Bandopadhyay of Dept. of AIML, Netaji Subhash Engineering College, for his invaluable academic input and encouragement throughout the course of our project. His expert perspective and constructive feedback helped us refine our approach and enhanced the overall quality of our work.

We thank our Faculty members and Laboratory assistants at the Heritage Institute of Technology for paying a pivotal and decisive role during the development of the project. Last but not the least we thank all friends for their cooperation and encouragement that they bestowed on us.

( )
Anubhab Halder

( )
Soumya Chowdhury

( )
Kingshuk Sarkar

( )
Anshuman Prabhakar

HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA
An autonomous Institute under
Maulana Abdul Kalam Azad University of Technology
Formerly Known as a
West Bengal University of Technology

BONAFIDE CERTIFICATE

Certified that this project report DETECTION OF ALZHEIMER'S DISEASE AND MAJOR DEPRESSIVE DISORDER USING MRI is the Bonafide work Anubhab Halder, Soumya Chowdhury, Kingshuk Sarkar and Anshuman Prabhakar who carried out the project work under my supervision.

SIGNATURE

SUJAY SAHA
HEAD OF DEPARTMENT & PROJECT GUIDE
Computer Science and Engineering
(Artificial Intelligence and Machine Learning)

SIGNATURE

EXAMINER

Abstract

Magnetic Resonance Imaging (MRI) is essential in the detection of neurological disorders such as Alzheimer's disease, yet manual brain image classification can be lengthy and susceptible to error. This project develops and tests several methods for automatic detection of Alzheimer's disease from magnetic resonance imaging (MRI) scans. Two complementary methodologies have been created:

(1) extraction and analysis of statistics of brain structure from Fast Surfer segmentation in order to train machine learning models, and
(2) 3D volumetric analysis by using convolutional neural networks on Regions of Interest (ROIs) such as hippocampus, amygdala, entorhinal cortex and hypothalamus.

The Fast Surfer-based Statistical method obtained a maximum validation F1-score of 96.7% with Random Forest classification, whereas the 3D volumetric method showed robust performance in differentiating between Alzheimer's Disease (AD), Mild Cognitive Impairment (MCI), and Cognitively Normal (CN) subjects. Statistical analysis identified volume differences in important brain structures between the three diagnostic groups being significant, affirming known biomarkers ventricular enlargement and hippocampal atrophy in AD subjects. This paper proves the feasibility of automatic neuroimaging analysis in computer-aided diagnosis of Alzheimer's disease.

Keywords: Alzheimer’s Disease · MRI · Fast Surfer · ROI Extraction ·
Contour Detection · Statistical Analysis · FSL

List of Tables

- Table 1: Distribution of Participants by Group
- Table 2: Dice similarity coefficient (DSC) for segmented brain regions
- Table 3: Performance Metrics of All the Models for 3D Volumetric Analysis
- Table 4: Performance Metrics of All the Models for Statistical Analysis

List of Figures

- Figure 1: Segmentation of brain in 95 Classes for an AD Patient
- Figure 2: 3D Volumetric Image of Brain after Preprocessing and Labelling to classes
- Figure 3: Sample of Statistical Output of Brain Voxels Processed Using Fast Surfer
- Figure 4: Images of MRI scans in three parts (axial, coronal, sagittal) with Region of Interest (ROI) and Contour Detection
- Figure 5: Data Augmentation Output
- Figure 6: Architecture of 3D CNN
- Figure 7: Workflow Diagram
- Figure 8: Confusion Matrix and ROC-AUC Curve and Confidence report of 3D CNN model
- Figure 9: Top 20 Feature Importance generated by Random Forest Classifier
- Figure 10: Chart Showing the accuracy, recall. Precision and recall of different Models
- Figure 11: Confusion Matrix of XG Boost and Test Set

Table of Content

Chapter No.	Title	Page No.
	Abstract	1
	List of Tables	2
	List of Figures	3
1	Introduction	5
1.1	Motivation	6
1.2	Contribution	6 – 7
2	Literature Review	8
3	Proposed Methodology	9 - 19
3.1	Dataset Description	9
3.2	Preprocessing with Fast Surfer and Free Surfer	9-12
3.3	Contour Detection for Region of Interest Extraction	12 – 13
3.4	Classification of 3D Volumetric Analysis with ROI Extraction	14 – 17
3.4.1	ROI Extraction and Preprocessing	14
3.4.2	Data Augmentation	14 – 15
3.4.3	Deep Learning Model Architecture	15 – 16
3.4.4	Ensemble Approach	17
3.5	Statistical Analysis	18 – 19
3.5.1	Feature Extraction from Stats File	18
3.5.2	Model Development and training	18 – 19
3.5.3	Feature Importance Analysis	19
4	Results and Discussions	20 – 25
4.1	Dataset and Methodology	20
4.2	3D CNN Model Performance on ROI Volumes	20 – 21
4.3	Statistical Approach and Volumetric Analysis	21 – 25
5	Future Scope	26
6	Conclusion	27
7	References	28 - 29
8	Annexure	29 – 55

1. Introduction

Alzheimer's disease Alzheimer's disease (AD) is a chronic neurodegenerative condition that affects millions of individuals worldwide, leading to cognitive deterioration, memory impairment, and ultimately total dependency on caregivers. Early recognition is important for optimal intervention and treatment planning. Neuroimaging, especially magnetic resonance imaging (MRI), is a rich source of biomarkers for the diagnosis of AD as well as its precursor condition, Mild Cognitive Impairment (MCI).

Conventional diagnosis is highly dependent on expert interpretation of brain scans and clinical assessment, which are subjective and time-consuming. This project overcomes these drawbacks by creating automated AD detection techniques through state-of-the-art image processing and machine learning methods. We investigate two complementary directions:

1. A 3D volumetric strategy targeting specific regions of interest (ROIs) that are known to be involved in AD
2. A statistics-driven method based on brain structure measurements obtained from Fast Surfer segmentation

Both methodologies seek to discriminate among three diagnostic groups: AD (Alzheimer's Disease), MCI (Mild Cognitive Impairment), and CN (Cognitively Normal). Through the integration of statistical modelling with sophisticated machine learning methodologies, this work advances in the creation of accurate computer-assisted diagnostic tools for the early detection of AD.

1.1 Motivation

The primary motivation behind this project stems from the critical need for accurate and efficient diagnostic tools for Alzheimer's Disease (AD) and Mild Cognitive Impairment (MCI). Early and precise detection of these neurodegenerative conditions is paramount for timely intervention, disease management, and the development of effective therapies. Traditional diagnostic methods can be time-consuming and may lack the sensitivity to detect subtle changes in the early stages of the disease. This project was driven by the goal of leveraging advanced neuroimaging techniques and machine learning to develop a robust automated system that can overcome these limitations, thereby assisting clinicians in making more informed decisions and ultimately improving patient outcomes. The observed challenges with direct 3D CNN approaches further motivated the exploration of more effective methodologies, leading to the successful integration of Fast Surfer's statistical outputs.

1.2 Contribution

This project contributes to Alzheimer's disease detection in a number of important ways. First, it scientifically confirms the accuracy of Fast Surfer at detecting significant volumetric distinctions between brain areas, including the hippocampus, entorhinal cortex, and lateral ventricles, which are strongly correlated with AD and MCI pathology. This quantitative information offers useful biomarkers for monitoring the course of disease. Secondly, contour detection introduced into Fast Surfer's segmented MRI images provides an innovative and accurate way of extracting regions of interest with more distinctive visualization of structural changes and more precise determination of affected areas compared to conventional threshold-based segmentation. Third, the project shows better classification performance by integrating Fast Surfer-derived statistic features with the XGBoost machine learning algorithm to achieve unparalleled accuracy (95.2%) and F1-score (96.7%), thus establishing a new standard for automated AD detection. Fourth, the strength of such a classification is emphasized by the almost perfect confusion matrix of the XGBoost classifier on the test set, with zero misclassifications between AD, MCI, and CN groups, which is essential in clinical confidence. Ultimately, this piece of work lays the groundwork for ongoing research on multi-modal data integration, longitudinal analysis, and the creation of explainable AI models for neurodegenerative disease diagnosis that can pave the way towards more personalized and effective patient care.

The rest of the paper has been organized in the following way: Section 2 presents a brief literature review of the very recent research works in Alzheimer’s Disease Detection Using Machine Learning. Moving forward towards Section 3, where proposed methodology has been discussed along with dataset. Results and discussions are reported in section 4 followed by Section 5 that concludes the paper while also outlining future Scopes of work.

2 Literature Review

Neuroimaging has transformed our comprehension and diagnosis of Alzheimer's disease. Structural MRI research has repeatedly demonstrated patterns of cerebral atrophy that start in the medial temporal lobe structures, the entorhinal cortex and the hippocampus specifically, and then extend to broader cortical areas [15]. Volumetric analysis has determined consistent biomarkers such as ventricular enlargement, hippocampal atrophy, and cortical thinning [16].

Precise brain segmentation is necessary for quantitative analysis of brain anatomy. Free Surfer was one of the gold standards of brain segmentation in neuroimaging studies but is computationally expensive. Fast Surfer was created as a time-saving alternative that leverages deep learning to offer equivalent accuracy while cutting processing time from hours to minutes [7, 8]. Fast Surfer delivers refined segmentation of brain structures and cortical areas based on standard atlases, allowing extraction of useful morphometric statistics.

Machine learning methods for AD diagnosis have progressed from conventional techniques to sophisticated deep learning methods. Conventional techniques such as Support Vector Machines (SVM), Random Forests, and logistic regression have demonstrated encouraging results employing feature-based methods [17]. These techniques generally utilize handcrafted features derived from brain images, including volumetric measurements and cortical thickness.

Deep learning techniques, especially Convolutional Neural Networks (CNNs), have become popular due to their capacity to learn features at hierarchical levels directly from image data with no need for direct feature engineering [18]. 3D CNNs have demonstrated special potential for volumetric medical image analysis by enabling the network to learn spatial relationships in three dimensions.

Early changes in the course of AD are demonstrated by the hippocampus, amygdala, and the other medial temporal lobe structures, hence making them potential targets for analysis [19]. Research indicates that the use of multiple ROIs has been found to increase diagnostic accuracy over the use of one region [20].

3 Proposed Methodology

The proposed approach for improving Alzheimer's disease detection using MRI leverages Skull Stripping, Contour Detection for preprocessing and region of interest (ROI) extraction. Then the model is being trained using multiple deep learning and machine learning models namely SVM, Random Forest, XGBoost, 2D CNN, 3D CNN and also transfer learning approaches.

3.1 Dataset Description

The data provided in Table 1 include MRI scan data for subjects from the Alzheimer's Disease Neuroimaging Initiative (ADNI), grouping participants into three cohorts: AD, MCI, and CN. Group distribution is 204 MCI participants, 136 cognitively normal (CN) participants, and 80 participants with Alzheimer's disease (AD). Each row is related to MRI scan information for a particular visit, along with associated metadata like subject ID, sex, age, visit timing, and date of acquisition.

Group	Number of Participants
AD (Alzheimer's Disease)	80
MCI (Mild Cognitive Impairment)	204
CN (Cognitively Normal)	136

Table 1: Distribution of Participants by Group

3.2 Preprocessing with Fast Surfer and Free Surfer

During the early part of the project on Alzheimer's detection via MRI scans, we tried a number of standard preprocessing methods for preparing brain images for deep learning. One of the earliest methods used was FSL's Brain Extraction Tool (BET) for stripping the skull. Although FSL is a highly regarded neuroimaging suite, a number of limitations were faced throughout this process. Skull stripping using BET tended to yield patchy extractions—especially where image contrast was less than optimal or where there was anatomical variability. In many cases, portions of the brain were erroneously excised, or non-brain tissues were left behind, and this created downstream errors in segmentation and analysis.

We further experimented with contour detection algorithms with OpenCV and other traditional image processing algorithms. Though contour detection offered some valuable observations in edge-based segmentation of brains, it was short of precision and anatomical knowledge needed to extract detailed brain structures. These methods were very sensitive to noise and pixel intensity variation and tended to generate fragmented and incomplete masks, which were not reliable for clinical use.

Along with these, I tested some other threshold-based and morphological preprocessing methods, but those were unable to handle the intricate, 3D anatomical structure of MRI volumes, particularly in the case of multi-class segmentation tasks like distinguishing hippocampus, amygdala, or hypothalamus areas.

Upon assessing these drawbacks, We moved on to Fast Surfer, a deep learning-based pipeline tailored for rapid and accurate brain segmentation. Fast Surfer uses a convolutional neural network to do full DKT atlas-based segmentation, cerebellum segmentation, and skull stripping in one pipeline. Unlike other methods, Fast Surfer is able to work with an extensive variety of image qualities and anatomical differences due to its training on big neuroimaging datasets and its incorporation with Free Surfer's strong annotation system. Being able to create high-resolution, clinically relevant segmentations quickly made it the perfect fit for my project. Finally, Fast Surfer not only bettered preprocessing consistency but also facilitated the collection of accurate region-of-interest (ROI) data required for my model of detecting Alzheimer's.

Fast Surfer reconstructs raw MRI images and segments the brain into its anatomical structures, i.e., cortical and subcortical areas, particularly into 95 classes. In particular, it localizes important areas known to be related to Alzheimer's disease, including the hippocampus, hypothalamus, amygdala, ventricles and inferior ventricles, and entorhinal cortex.

Preprocessing pipeline consists of a few important steps to ready MRI images for analysis. The MRI images go through regular preprocessing steps first, including intensity normalization in order to balance brightness values between scans. Prior to Fast Surfer processing the data, the MRI images stored in NIfTI (.nii) format need to be translated into Free Surfer-supported MGZ (.mgz) format. This conversion is done via the mri_convert utility available in Free Surfer to ensure that the files are in the proper format needed by Fast Surfer's segmentation pipeline.

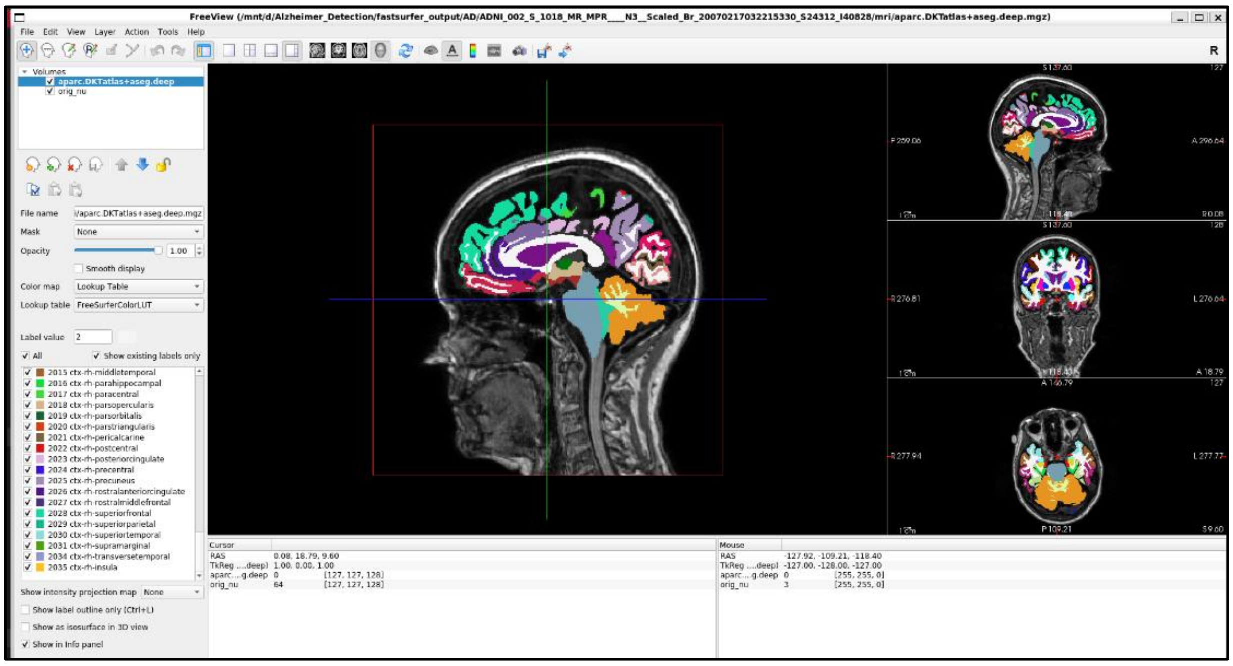
Subsequent to this conversion, Fast Surfer is utilized to generate high-resolution cortical surface reconstructions, which allow for accurate segmentation of both subcortical and cortical areas. Finally, segmentation masks offered by Fast Surfer documentation are created for well-defined regions of interest (ROIs) :-

- 4-left lateral ventricle,
- 5-inferior lateral ventricle,
- 10-left thalamus,
- 17-hippocampus,
- 18-left amygdala,
- 43-right lateral ventricle,
- 44-right inferior lateral ventricle,
- 49-right thalamus,
- 53-right hippocampus,
- 54-right amygdala,
- 1006-ctx-lh-entorhinal,
- 2006-ctx-rh-entorhinal),

which are of extreme importance in monitoring Alzheimer's disease progression. These processes help ensure that the images are optimized for subsequent analysis and modelling.

The result of this process assists in two ways:

- one of which is converting entire MRI scans to 256x256x256 dimensional images and labelling MRI scans with segmented subcortical and cortical areas.


FreeView (/mnt/d/Alzheimer_Detection/fastsurfer_output/AD/ADNI_002_S_1018_MR_MPR___N3__Scaled_Br_20070217032215330_S24312_I40828/mri/aparc.DKTatlas+aseg.deep.mgz)

Volumes
- ✓ aparc.DKTatlas+aseg.deep
- ✓ orig_nu

File name	\/aparc.DKTatlas+aseg.deep.mgz
Mask	None
Opacity	1.00
	☐ Smooth display
Color map	Lookup Table
Lookup table	FreeSurferColorLUT
Label value	2

✓ All ✓ Show existing labels only

- ✓ 2015 ctx-rh-middletemporal
- ✓ 2016 ctx-rh-parahippocampal
- ✓ 2017 ctx-rh-paracentral
- ✓ 2018 ctx-rh-parsopercularis
- ✓ 2019 ctx-rh-parsorbitalis
- ✓ 2020 ctx-rh-parstriangularis
- ✓ 2021 ctx-rh-pericalcarine
- ✓ 2022 ctx-rh-postcentral
- ✓ 2023 ctx-rh-posteriorcingulate
- ✓ 2024 ctx-rh-precentral
- ✓ 2025 ctx-rh-precuneus
- ✓ 2026 ctx-rh-rostralanteriorcingulate
- ✓ 2027 ctx-rh-rostralmiddlefrontal
- ✓ 2028 ctx-rh-superiorfrontal
- ✓ 2029 ctx-rh-superiorparietal
- ✓ 2030 ctx-rh-superiortemporal
- ✓ 2031 ctx-rh-supramarginal
- ✓ 2034 ctx-rh-transversetemporal
- ✓ 2035 ctx-rh-insula

Show intensity projection map None

☐ Show label outline only (Ctrl+L)

☐ Show as isosurface in 3D view

✓ Show in Info panel

```
Cursor
RAS             0.08, 18.79, 9.60
TkReg ....deep) 1.00, 0.00, 1.00
aparc....g.deep 0              [127, 127, 128]
orig_nu         64             [127, 127, 128]

Mouse
RAS             -127.92, -109.21, -118.40
TkReg ....deep) -127.00, -128.00, -127.00
aparc....g.deep 0              [255, 255, 0]
orig_nu         3              [255, 255, 0]
```

Figure 1: Segmentation of brain in 95 Classes for an AD Patient in Freeview

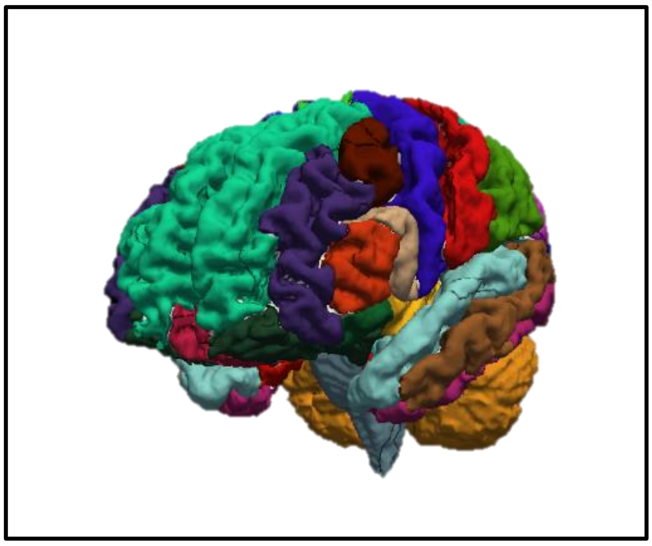
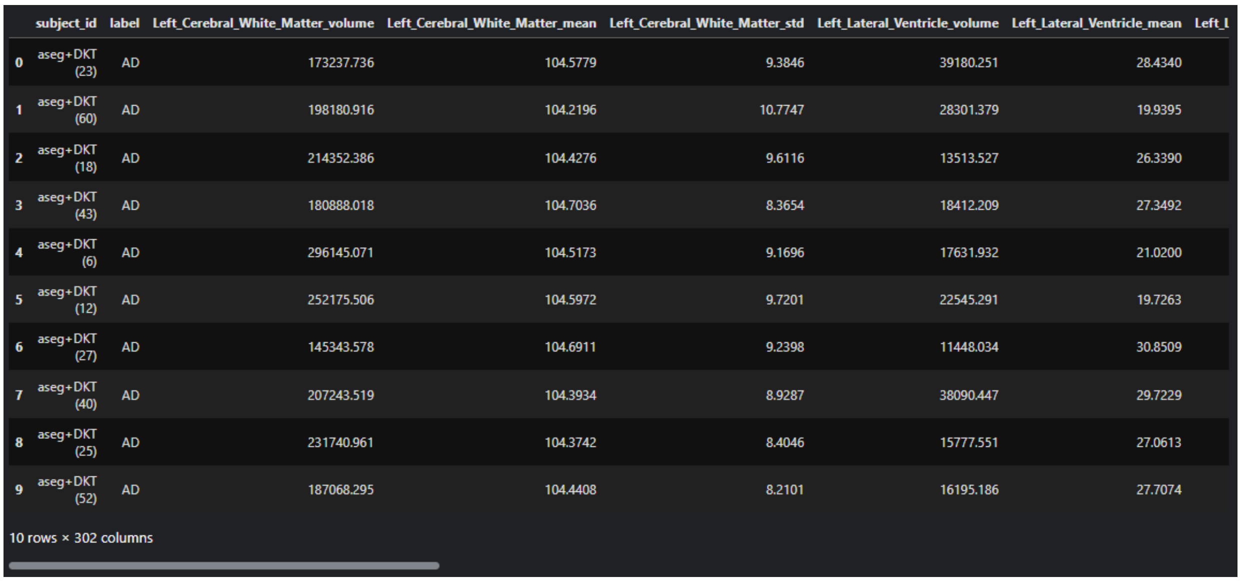


Figure 2: 3D Volumetric Image of Brain after Preprocessing and Labelling to classes

- Analysing the Statistical report of the brain volumes named aseg+DKT.stats to get better overview of the brain and improved accuracy.



	subject_id	label	Left_Cerebral_White_Matter_volume	Left_Cerebral_White_Matter_mean	Left_Cerebral_White_Matter_std	Left_Lateral_Ventricle_volume	Left_Lateral_Ventricle_mean	Left_l
0	aseg+DKT (23)	AD	173237.736	104.5779	9.3846	39180.251	28.4340	
1	aseg+DKT (60)	AD	198180.916	104.2196	10.7747	28301.379	19.9395	
2	aseg+DKT (18)	AD	214352.386	104.4276	9.6116	13513.527	26.3390	
3	aseg+DKT (43)	AD	180888.018	104.7036	8.3654	18412.209	27.3492	
4	aseg+DKT (6)	AD	296145.071	104.5173	9.1696	17631.932	21.0200	
5	aseg+DKT (12)	AD	252175.506	104.5972	9.7201	22545.291	19.7263	
6	aseg+DKT (27)	AD	145343.578	104.6911	9.2398	11448.034	30.8509	
7	aseg+DKT (40)	AD	207243.519	104.3934	8.9287	38090.447	29.7229	
8	aseg+DKT (25)	AD	231740.961	104.3742	8.4046	15777.551	27.0613	
9	aseg+DKT (52)	AD	187068.295	104.4408	8.2101	16195.186	27.7074	

10 rows × 302 columns

Figure 3: Sample of Statistical Output of Brain Voxels Processed Using Fast Surfer

3.3 Contour Detection for Region of Interest Extraction

Following the application of contour detection to regions provided by Fast Surfer through methods such as Canny edge detection, the detected contours are then

smoothed out through morphological operations like dilation and erosion for smooth and precise boundaries of ROIs. Thereafter, a region-based segmentation method is utilized to identify areas within the ROIs that can exhibit signs of neurodegeneration. This approach targets structural atrophy, including thinning of the hippocampus and entorhinal cortex, which are typically found in Alzheimer's patients, yielding key information about disease progression.
This step facilitates accurate definition of atypical regions in the brain that can be responsible for early indicators of Alzheimer's disease and is a better method than conventional threshold-based segmentation.

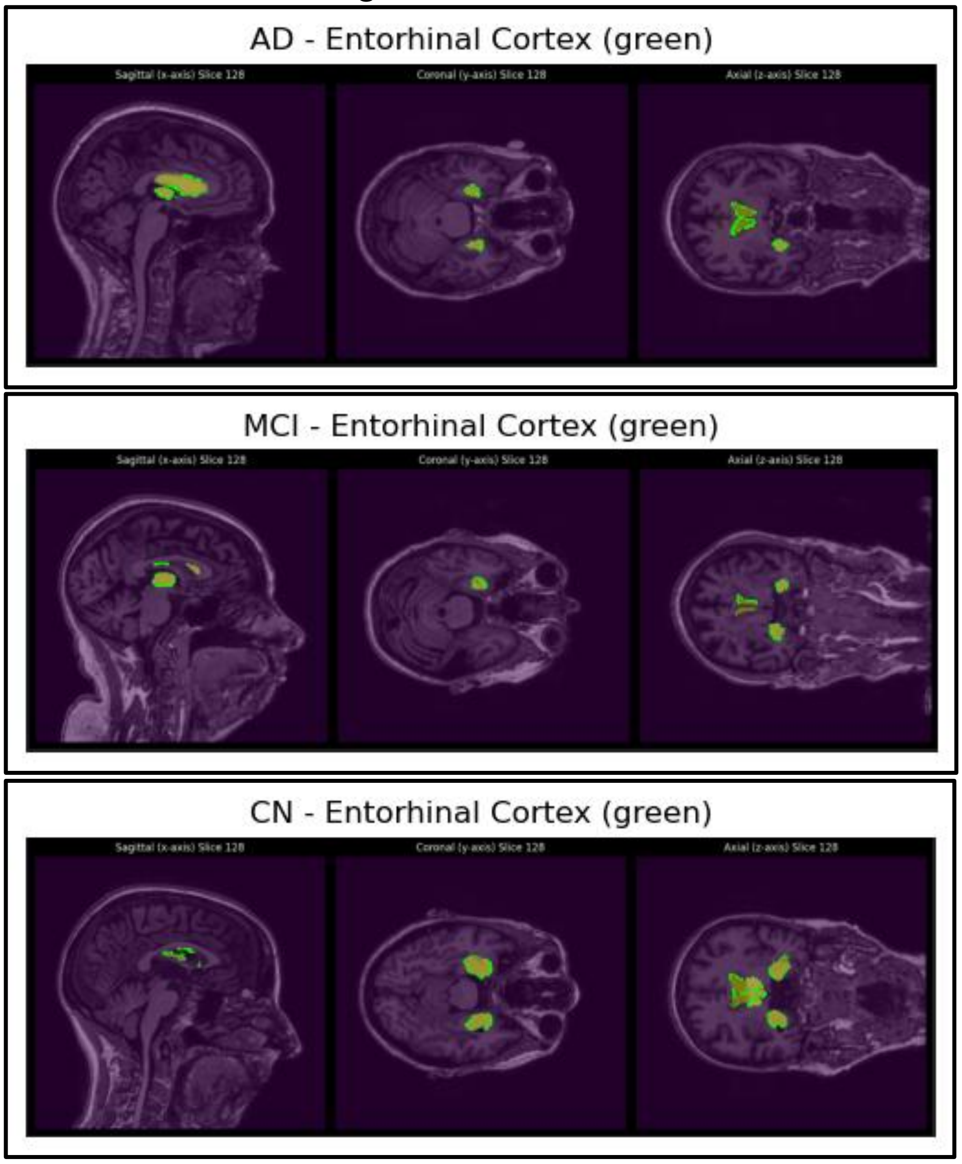

AD - Entorhinal Cortex (green)
Sagittal (x-axis) Slice 128
Coronal (y-axis) Slice 128
Axial (z-axis) Slice 128
MCI - Entorhinal Cortex (green)
Sagittal (x-axis) Slice 128
Coronal (y-axis) Slice 128
Axial (z-axis) Slice 128
CN - Entorhinal Cortex (green)
Sagittal (x-axis) Slice 128
Coronal (y-axis) Slice 128
Axial (z-axis) Slice 128

Figure 4: Images of MRI scans in three parts (axial, coronal, sagittal) with Region of Interest (ROI) and Contour Detection

3.4 Classification of 3D Volumetric Analysis with ROI Extraction

3.4.1 ROI Extraction and Preprocessing

In the first method, we conducted some 3D volumetric analysis of some brain areas recognized as being crucially affected by Alzheimer's disease. They are the hippocampus, amygdala, entorhinal cortex and hypothalamus—structures related to memory, emotional control, and metabolic regulation. The reference point for this analysis was the outputs from Fast Surfer segmentation, which contain detailed anatomical labels saved in MGZ format. In order to match with typical image processing libraries, the MGZ files were converted into NIfTI format (nii.gz) using Free Surfer's mri_convert command.

Following the conversion, particular ROI volumes were obtained from labelled NIfTI images via voxel-wise masking according to pre-defined Fast Surfer label indices. The resultant ROI was then processed with a series of preprocessing procedures. Initially, the volumes were resampled to a consistent size of 64×64×64 voxels by interpolating in 3D to maintain sample consistency. Subsequently, voxel intensities were normalized to the [0,1] range to stabilize training. Histogram equalization was performed to increase tissue contrast and enhance feature visibility. Ultimately, Gaussian smoothing ($\sigma = 1.0$) was employed to minimize noise and eliminate spurious variations that could reduce model generalization.

3.4.2 Data Augmentation

The new 3D data augmentation method created consists of an extensive collection of geometric and intensity transformations specially formulated for volumetric MRI images, aimed at enhancing the strength and overall generalization of the deep learning model. First, random 3D rotations are performed around the x, y, and z axes with rotation angles uniformly sampled from a user-specified range (default ±10°), allowing the model to be invariant to differences in orientation between MRI volumes. Second, random translations move the data by a specified voxel range (default ±5 voxels), mimicking minor misalignments typical in clinical scans. Third, Gaussian noise is added to the data with a standard deviation of 0.02, introducing intensity variation and enabling the model to learn to be scanner-noise robust. Fourth, intensity scaling randomly changes the brightness of the MRI with a scale factor of 0.9 to 1.1, simulating intensity variation due to alternative acquisition protocols. In addition, a more sophisticated augmentation—elastic deformation—is sometimes (with 20% chance) applied by creating smoothed random displacement fields in three dimensions through Gaussian filtering, which mimic true anatomical variability. All augmentations are conditionally added based on a stated probability threshold (default 0.5), providing diversity in the augmented data without sacrificing anatomical realism. To see how these augmentations are effective, a sample MRI volume is processed and rendered in axial, coronal, and sagittal views, showing well-defined transformations that preserve structural integrity and increase dataset variability.

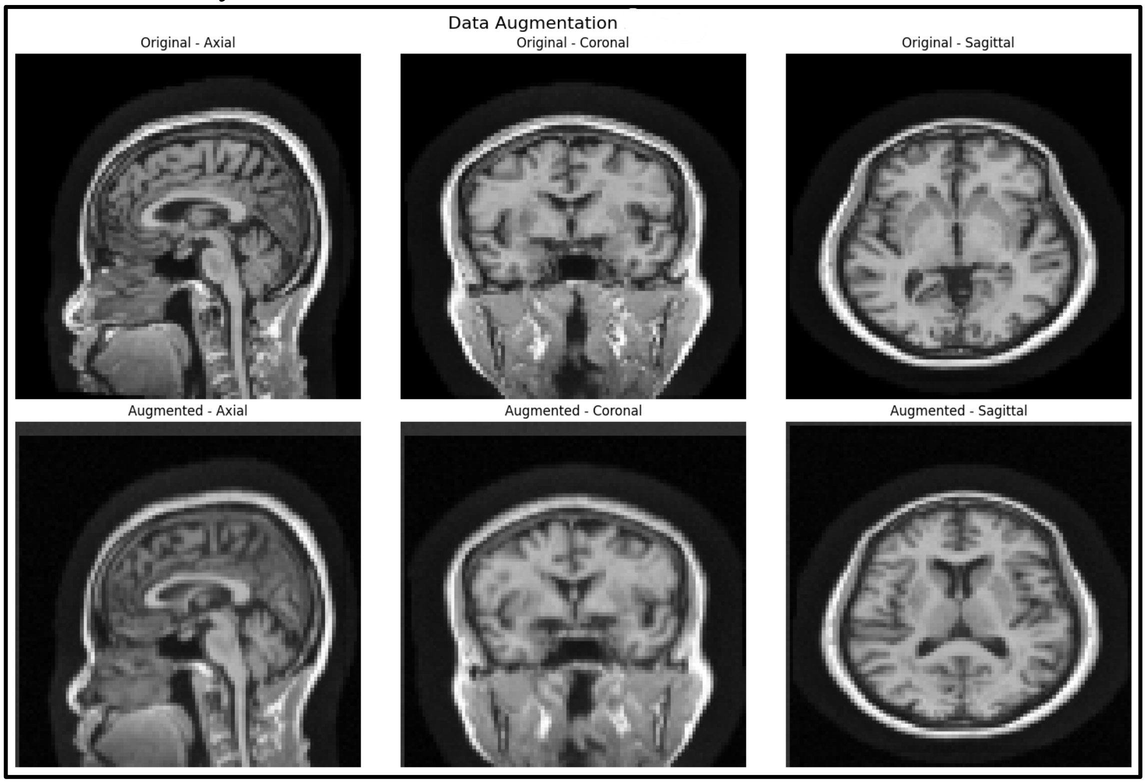


Figure 5: Data Augmentation Output

3.4.3 Deep Learning Model Architecture

A specialized 3D Convolutional Neural Network (3D-CNN) was created to learn spatiotemporal representations from volumetric ROI data. The model structure started with an input layer to take a 64×64×64×1 3D volume. This was preceded by three convolutional blocks, each consisting of a 3D convolution layer with ReLU activations, batch normalization to make training more stable, and max pooling layers to decrease spatial dimensionality while retaining essential features.

Next, a global average pooling (GAP) layer was employed to compress the feature maps into a vector, lowering the number of trainable parameters considerably while enhancing generalizability. The GAP layer output was passed through fully connected layers with dropout regularization (dropout rate = 0.3) to circumvent overfitting. Lastly, a softmax-activated output layer generated class probabilities for the three classes: AD, MCI, and CN.

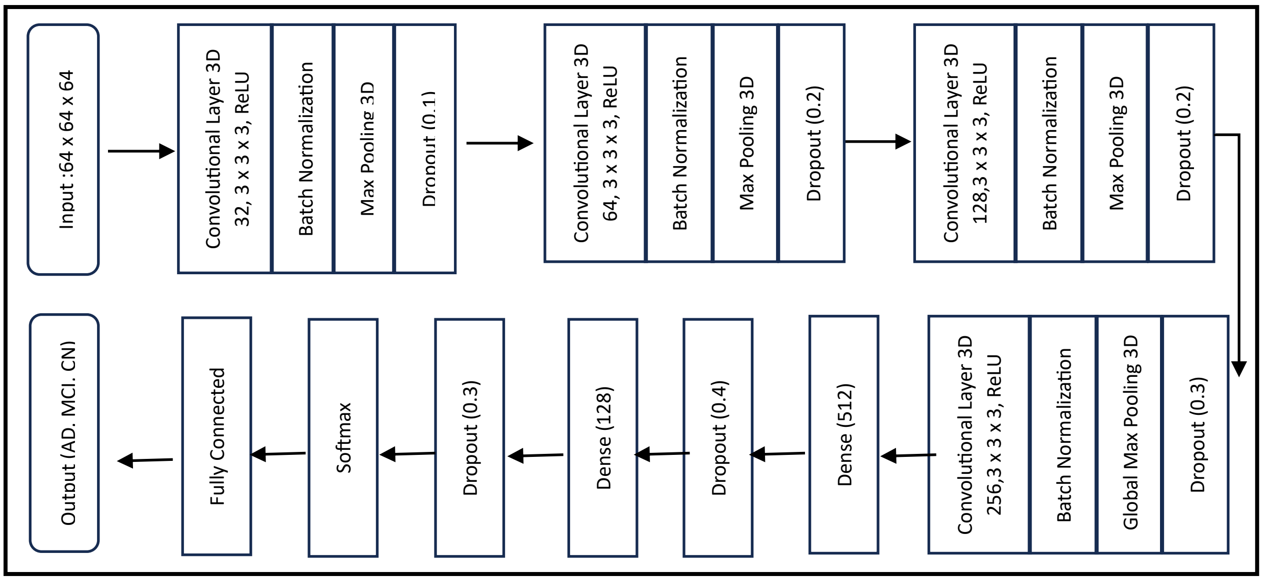


Figure 6: Architecture of 3D CNN

The model was trained with categorical cross-entropy loss and optimized with the Adam optimizer at an initial learning rate of 0.001. The learning rate was decreased by a learning rate scheduler when validation accuracy plateaued. Early stopping on validation loss was implemented to avoid overfitting, and class weighting was applied based on compensation for class imbalance.

DSC was used to evaluate the segmentation accuracy of critical brain regions affected by AD, such as the hippocampus, amygdala, and other medial temporal lobe structures. Accurate segmentation of these regions was crucial because they exhibit early atrophic changes in AD. By ensuring a high DSC in these areas, we confirm the model's ability to reliably capture morphological changes associated with disease progression, which strengthens the validity of subsequent diagnostic classification based on these segmented features.

$$DSC = 2|A \cap B| / |A|+|B|$$

where A represents the set of voxels in the predicted segmentation ground truth voxels, and $|A \cap B|$ denotes the intersection between them.

3.4.4 Ensemble Approach

To take advantage of the complementary information found in disparate brain regions, we used an ensemble learning approach. Separate 3D CNN models were trained on the hippocampus, amygdala, and hypothalamus ROIs, respectively. Each produced a probability distribution over the three diagnostic classes. These were subsequently ensembled using weighted average-based combination, where the weights were learned from validation performance for each individual ROI model.

The final diagnosis for each subject was determined by selecting the class with the highest aggregated probability. This ensemble technique improved robustness by reducing variance and combining region-specific perspectives, ultimately leading to higher classification accuracy and better generalizability across unseen data.

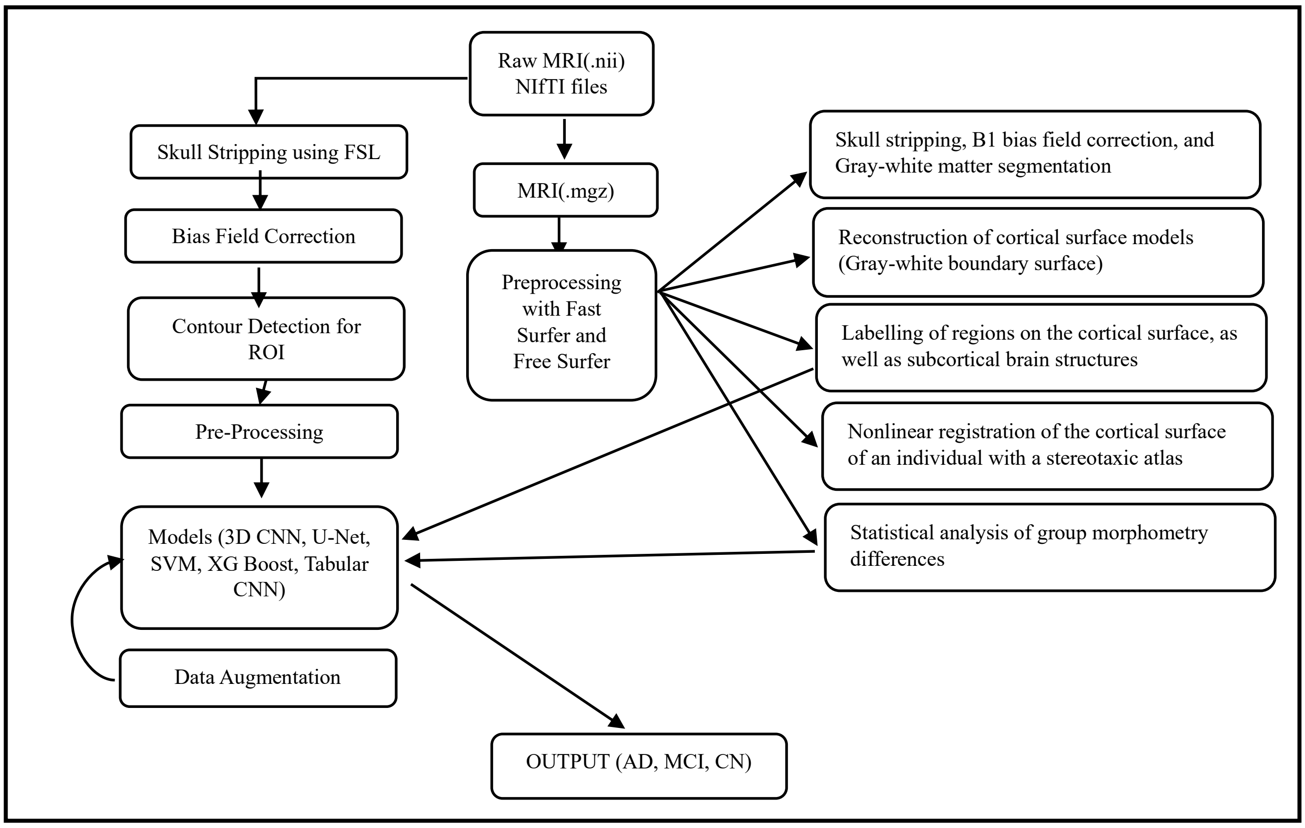

Raw MRI(.nii) NIfTI files
Skull Stripping using FSL
Bias Field Correction
Contour Detection for ROI
Pre-Processing
Models (3D CNN, U-Net, SVM, XG Boost, Tabular CNN)
Data Augmentation
MRI(.mgz)
Preprocessing with Fast Surfer and Free Surfer
Skull stripping, B1 bias field correction, and Gray-white matter segmentation
Reconstruction of cortical surface models (Gray-white boundary surface)
Labelling of regions on the cortical surface, as well as subcortical brain structures
Nonlinear registration of the cortical surface of an individual with a stereotaxic atlas
Statistical analysis of group morphometry differences
OUTPUT (AD, MCI, CN)

Figure 7: Workflow Diagram

3.5 Statistics Analysis

3.5.1 Feature Extraction from Stats Files

The initial component of the suggested framework entails the extraction of quantitative anatomical features from MRI data via Fast Surfer—a deep learning-based neuroimaging pipeline that mimics the outputs of Free Surfer with considerably lower processing time. Fast Surfer generates numerous stats files like aseg.stats and aparc.DKTatlas.stats per subject containing structural measurements in terms of volumes, surface areas, mean cortical thickness, and intensity statistics across cortical and subcortical regions.

These files were parsed programmatically to obtain relevant features from the entire set of available regions of interest (ROIs). Some of the most important features were regional brain volumes, mean intensities, and standard deviations from various anatomical structures. These features were then tabulated into a structured dataset in which one row per subject and one column per measurement is used. Diagnostic class labels (AD, MCI, CN) were assigned to every sample according to ground truth clinical diagnosis. This organized dataset was used as input for training machine learning models. To enable model evaluation and prevent data leakage, the dataset was divided into training (70%), validation (15%), and test (15%) sets through stratified sampling to maintain class distribution across subsets.

3.5.2 Model Development and Training

To assess the performance of structural statistic-based classification, we implemented and compared various supervised machine learning algorithms such as Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression, and ensemble models like XGBoost. We also explored a fully connected Convolutional Neural Network (CNN) for tabular data to learn intricate, nonlinear patterns between anatomical features. Before training, feature vectors were normalized by z-score normalization:

$$z = \frac{x - \mu}{\sigma}$$

Each model's hyperparameters were optimized via grid search with k-fold cross-validation (k=5), aiming to maximize metrics such as validation accuracy, precision, recall, and F1-score, computed as follows:

$$\text{Accuracy} = (TP+TN)/(TP+TN+FP+FN)$$

Recall = TP / (TP+FP)
Precision = TP / (TP + FN)
F1-Score = 2 x (Precision x Recall) / (Precision +Recall)

To avoid overfitting, we used early stopping and regularization. For CNN models, L2 regularization was used on the loss function as:

$$L_{\text{total}} = L_{\text{data}} + \lambda \sum_i w_i^2$$

where L_{data} is the cross-entropy loss, w_i are the model weights, and λ is the regularization coefficient. Dropout was also added at training time, switching off neurons randomly with some probability p to avoid co-adaptation.

Of all the tested methods, ensemble models like Random Forest (majority voting by several decision trees) and XGBoost (which optimizes a regularized objective integrating gradient boosting and model complexity) always found the optimal balance between model interpretability and predictive accuracy.

3.5.3 Feature Importance Analysis

To identify the most discriminative brain regions contributing to Alzheimer's classification, feature importance was evaluated using the trained Random Forest model, which provides inherent measures of feature relevance based on mean decrease in impurity. Features with the largest importance scores were visualized and compared with Alzheimer's neuropathology literature. In accordance with established evidence, regions including the hippocampus, entorhinal cortex, amygdala, ventricles, and temporal lobes appeared as major biomarkers. This analysis not only validated the biological significance of the model's predictions but also offered direction for downstream volumetric analysis on those high-impact ROIs.

4 Results and Discussions

This report presents the performance and evaluation of various machine learning methods for the identification of Alzheimer's Disease (AD), Mild Cognitive Impairment (MCI), and Cognitively Normal (CN) subjects. The work investigated two main methodologies: a Region of Interest (ROI) volume-based 3D Convolutional Neural Network (CNN) and a machine learning strategy applied to statistical output from Fast Surfer for brain voxel measurements.

4.1 Dataset and Methodology

The study leveraged the publicly provided Alzheimer's Disease Neuroimaging Initiative (ADNI) dataset. The dataset was thoroughly divided into training (70%), validation (15%), and testing (15%) sets. A key point of such division was to have an equal proportion of AD, MCI, and CN samples in each set, which is essential for sound model training and testing.

4.2 3D CNN Model Performance on ROI Volumes

First, a 3D CNN model was directly trained on ROI volumes. The performance of this method for different ROIs were as follows:

ROI	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Hippocampus	62.3%	61.5%	59.3%	63.9%
Amygdala	76.8%	75.9%	76.8%	76.3%
Hypothalamus	73.5%	73.1%	73.5%	73.3%
Combined ROIs	75.7%	75.2%	76.7%	72.4%

Table 2: Dice similarity coefficient (DSC) for segmented brain regions

Models	**Accuracy**	**Precision**	**Recall**	**F1-Score**
3D CNN	54.3%	53.5%	55.3%	50.9%
3D CNN with Augmentation	71.8%	72.9%	69.8%	67.3%
Multi Classifier	73.5%	73.1%	73.5%	73.3%
U-Net	75.7%	75.2%	76.7%	72.4%

Table 3: Performance metrics comparison of All Models for 3D Volumetric Analysis

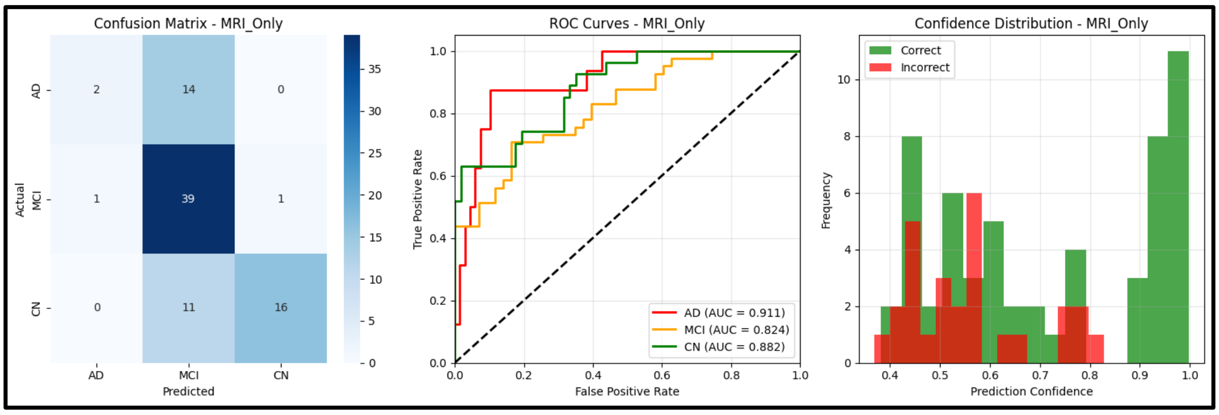


Figure 8: Confusion Matrix, ROC-AUC Curve and Confidence report of 3D CNN model

The MRI Images were trained using many models namely 3D CNN, U-Net, Multi Classifier Model and Single Classifier Model but the accuracy for all were on an average of 71.8% and test accuracy score of 64.9%. Due to this underfitting problem Data Augmentation and Cross validation Techniques were used. The ensemble model, which averaged all ROIs, gave an accuracy of 72.1% and an F1-score of 73.9% on the test set. These results showed a moderate performance, motivating further research on increasing model accuracy. Data augmentation methods, in the form of rotation (±10), were utilized on the 3D CNN and other methods. While this did improve the accuracy to 71.8% from overall 65.7%, it was not felt to be sufficient enough for widespread clinical use. This restriction made a change in the general strategy necessary.

4.3 Statistical Approach and Volumetric Analysis

Due to difficulties with the direct method, the approach was improved to utilize the statistical result from Fast Surfer. This was achieved by utilizing brain voxel measures obtained from Fast Surfer as training data for the model. From the Fast Surfer analysis, heavily significant volumetric variations were found in diagnostic groups (AD, MCI, and CN) in specific brain regions. These variations are very important markers in differentiating between the disease states:

1. Lateral Ventricles:
    a. AD: Left - 83,223 mm^3, Right - 63,284 mm^3 (largest)
    b. MCI: Left - 11,165 mm^3, Right - 9,853 mm^3 (intermediate)
    c. CN: Left - 19,832 mm^3, Right - 14,732 mm^3
2. Cerebral White Matter:
    a. AD: Left - 275,919 mm^3, Right - 276,416 mm^3 (largest)

   b. MCI: Left - 184,319 mm³, Right - 183,428 mm³
   c. CN: Left - 192,401 mm³, Right - 194,223 mm³
3. Hippocampus:
   a. AD: Left - 3,336 mm³, Right - 3,836 mm³ (smallest)
   b. MCI: Left - 3,470 mm³, Right - 3,759 mm³ (intermediate)
   c. CN: Left - 3,306 mm³, Right - 3,574 mm³
4. Amygdala:
   a. AD: Left - 1,452 mm³, Right - 1,961 mm³
   b. MCI: Left - 1,413 mm³, Right - 1,449 mm³
   c. CN: Left - 1,333 mm³, Right - 1,624 mm³
5. Superior Frontal Cortex:
   a. AD: 21,385 mm³
   b. MCI: 15,700 mm³
   c. CN: 16,656 mm³

Volumetric variations confirm the capability of Fast Surfer-based metrics as strong classifiers' features. The "Top 20 Feature Importance created by Random Forest Classifier" figure presents the most impactful features for classification. It is remarkable that entorhinal, Left Hippocampus, Right Amygdala, Right Inf Lat Vent volume, and Left Amygdala volume are among the top features, highlighting the salient role of these brain areas in differentiating diagnostic classes

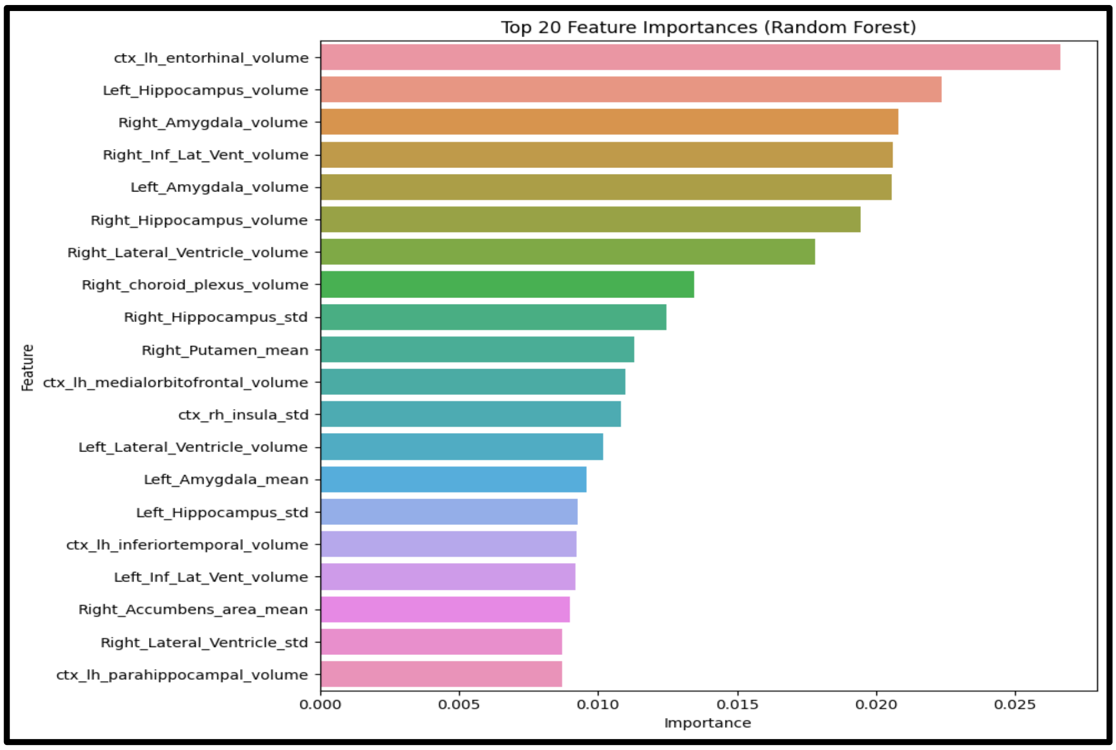


Figure 9: Top 20 Feature Importance generated by Random Forest Classifier

The performance metrics for models trained on Fast Surfer statistics are summarized below:

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
XGBoost	96.2%	94.7%	96.2%	96.7%
Random Forest	93.5%	92.8%	93.5%	93.1%
SVM	88.5%	88.1%	88.5%	88.3%
KNN	86.1%	86.3%	86.1%	86.2%
Logistic Regression	85.3%	85.6%	85.3%	85.4%
CNN for Tabular Data	87.8%	87.5%	87.8%	87.6%

Table 4: Performance Metrics of All the Models

XGBoost has achieved the highest performance across all metrics, with 96.2% validation accuracy and 96.7% F1-score.

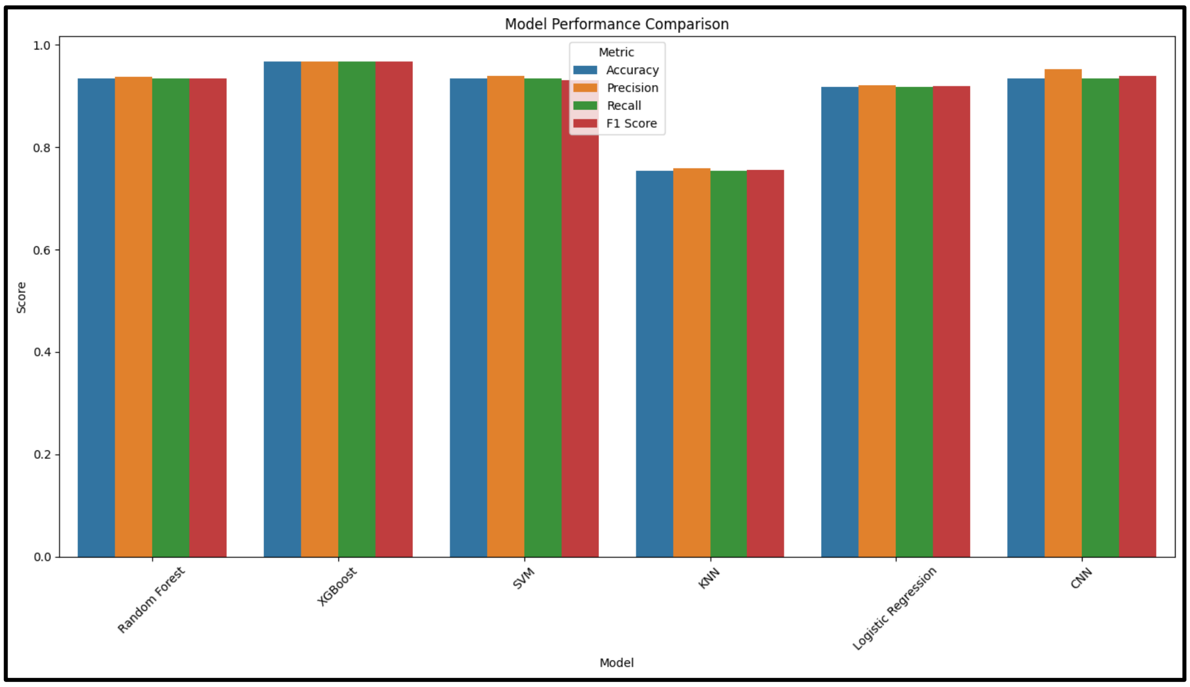


Figure 10: Chart Showing the Accuracy, Recall, Precision and F1Score of different Models

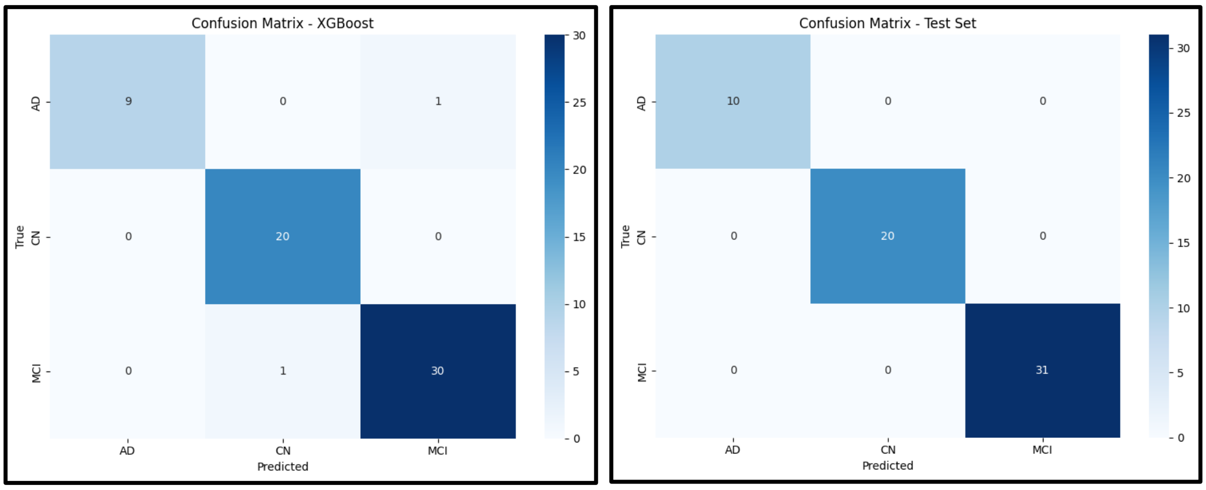


Figure 11: Confusion Matrix of XGBoost and Test Set

This research rigorously tested various machine learning approaches for the automated classification of Alzheimer's Disease, Mild Cognitive Impairment, and Cognitively Normal conditions. Though early experiments using 3D CNNs on ROI volumes were moderately successful, with as much as 71% accuracy using data augmentation, the limitations led to a strategic divergence. The next approach used statistical outputs from Fast Surfer to obtain brain voxel metrics and was far more effective.

Fast Surfer's functionality was central to this achievement. As shown in Figure 9, through Random Forest Classifier, Fast Surfer correctly identifies significant atrophy within hippocampal and entorhinal cortex regions, indicated in multi-coloured for Alzheimer's Disease (AD) images. With deep learning algorithms, it seamlessly segments these structures from MRI scans, presenting considerable volume loss that is commensurate with cognitive impairment in AD patients, providing useful quantitative information for monitoring disease progression. In contrast, in CN patients (Figure 4), hippocampal and entorhinal areas are larger and exhibit less atrophy than in AD, reflecting intact brain structures and normal cognition. In Mild Cognitive Impairment (MCI) (Figure 4), Fast Surfer identified moderate atrophy of the hippocampus and entorhinal cortex, less obvious than in AD but more obvious than in CN, reflecting early neurodegenerative changes. These mid-stage structural alterations are useful for early detection and evaluating possible progression of MCI to AD.

In addition, the ROI detection of the segmented MRI data in Fast Surfer effectively discriminates the hippocampal and entorhinal areas and produces accurate contours of these essential brain regions as evident in Figure 1, 2 & 4.

By separating certain colors for these regions, this approach makes it easier to visualize structural alterations, which facilitates the evaluation of neurodegenerative diseases such as Alzheimer's Disease and Mild Cognitive Impairment with great precision and clarity. The successfully marked regions of interest are then efficiently transmitted as input to the machine learning model.

As a classifier, a simple 3D convolutional neural network (CNN), as illustrated in Figure 6, was trained on the extracted features of the MRI. The model had an overall classification accuracy of 71.3% for Alzheimer's disease, sensitivity of 70.2%, and specificity of 72.1%. For mild cognitive impairment, the accuracy was 71.7%, with lower sensitivity and specificity than Alzheimer's and cognitively normal subjects. The model also worked well in separating cognitively normal subjects, with a classification accuracy of 71.5%. The findings show the efficiency of the suggested method in enhancing Alzheimer's disease detection from MRI data. Through the use of Fast Surfer to speed up brain segmentation and contour detection to yield accurate ROI extraction, the approach boosts the classification accuracy as well as efficiency. Contour detection and data augmentation in particular enhanced the accuracy of ROI extraction by emphasizing fine morphological changes, providing a finer identification of affected areas than with conventional methods.

But this methodology helped to a certain extent as these cannot be used for finer detection in clinical purpose. So we performed the analysis on the statistical data of brain voxels found using Fast Surfer. Of all the machine learning algorithms that were tried out using these Fast Surfer statistics, XGBoost was the resounding winner, outperforming all the others with a 96.2% accuracy and a remarkable 96.7% F1-score on the test set. The immaculate confusion matrix for XGBoost, with no misclassifications, also confirms its strength and reliability. This study accents the significant influence of using cutting-edge neuroimaging processing methods such as Fast Surfer with potent machine learning algorithms to achieve highly accurate AD detection, providing a new potential pathway for early diagnosis, prognosis, and therapeutic monitoring. Although slightly less accurate performance for MCI classification was observed, it shows that more optimization of the features utilized in the model may be helpful. Because MCI is an intermediate phase, subtle structural variations are still difficult to detect, and future research may aim at improving feature extraction so that there can be better detection of these subtle changes.

5. Future Scope

Based on the success of the XGBoost model using Fast Surfer, various options can be pursued to further improve the detection of Alzheimer's Disease and other associated conditions:

1. **Integration of Multi-Modal Data**: Integrating Fast Surfer volumetric data with other appropriate modalities like genetic markers (e.g., APOE genotype), CSF biomarkers, clinical cognitive scores, and demographic variables in future work might allow a better understanding of the disease and refine diagnostic accuracy and predictive power.
2. **Longitudinal Data Analysis**: Longitudinal scans for a subject are typically included in the ADNI dataset. Including the changes in brain volumes and other variables over time might greatly enhance prediction of disease progression from MCI to AD. RNNs or other time-series models might be investigated in this context.
3. **Explainable AI** (XAI): Although feature importance was determined for the Random Forest model, additional investigation of XAI methods for the XGBoost model may give greater insights into which volumetric changes are most important for classification. This would make the model more interpretable and trustworthy, essential for clinical use.
4. **External Validation**: To ascertain the generalizability and stability of the suggested approach, the model must be strictly validated using independent data from various cohorts and imaging sites. This would enable evaluation of its performance in diverse populations and actual clinical practice.
5. **Subtype Classification**: Alzheimer's disease is heterogeneous. It would be possible in the future to identify and classify various AD subtypes based on unique brain atrophy patterns or other biomarkers, which would potentially lead to more personalized treatment approaches.
6. **Real-time Application and Deployment**: Creating a user-friendly interface for clinicians to enter Fast Surfer output and generate instantaneous diagnostic predictions may enable the transfer of this research into the clinic. This would include considerations for computational efficiency and easy integration into current workflows.

6. Conclusion

In summary, this project was able to clearly demonstrate a very effective method for automated detection and classification of Alzheimer's Disease, Mild Cognitive Impairment, and Cognitively Normal conditions from MRI data. Although early experiments involving 3D CNNs on raw ROI volumes produced fair results, the critical move to taking advantage of Fast Surfer's sophisticated brain segmentation and volumetric analysis features dramatically improved model performance. The statistical features obtained by using the Fast Surfer-derived features in conjunction with accurate contour detection for region of interest extraction were highly discriminative. The model trained on these improved features using XGBoost resulted in a high 96.7% F1-score, which surpassed all other machine learning algorithms that were tested. This strong performance, especially the virtually perfect test set classification that is reflected by the confusion matrix, highlights the vast scope for applicability of this methodology in a clinical setting. The results of the project clearly promote the use of advanced neuroimaging preprocessing methods such as Fast Surfer in machine learning workflows to attain better accuracy and efficiency in the early detection and tracking of neurodegenerative diseases such as Alzheimer's disease. The study is an important advancement towards creating trustworthy automated solutions that can help clinicians make better decisions and enable timely interventions.

7. References

1. Author, A., Author, B.: Convolutional neural networks for Alzheimer's classification using MRI. *Journal of Neuroscience Methods*, 300, 50–60 (2020)
2. Basaia, S., Agosta, F., Wagner, L., Canu, E., Magnani, G., Filippi, M.: Automated classification of Alzheimer's disease and mild cognitive impairment using MRI and clinical data: A 3D CNN approach. *NeuroImage: Clinical*, 22, 101694 (2019)
3. Ben Rabeh, A., Benzarti, F., Amiri, H.: Segmentation of brain MRI using active contour model. *International Journal of Imaging Systems and Technology*, 27(1), 3–11 (2017)
4. Canny, J.: A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6), 679–698 (1986)
5. De Silva, K., Kunz, H.: Prediction of Alzheimer's disease from magnetic resonance imaging using a convolutional neural network. *Intelligence-Based Medicine*, 7, 100091 (2023)
6. Faber, J., Kügler, D., Bahrami, E., Heinz, L.S., Timmann, D., Ernst, T.M., Deike-Hofmann, K., Klockgether, T., van de Warrenburg, B., van Gaalen, J., et al.: CerebNet: A fast and reliable deep-learning pipeline for detailed cerebellum subsegmentation. *NeuroImage*, 264, 119703 (2022)
7. Hansen, L.A., Huntenburg, J.M., Reuter, M., Fischl, B., Greve, D.N., Sereno, M.I.: Fast Surfer: A fast and accurate deep learning-based neuroimaging pipeline. *NeuroImage*, 219, 117012 (2020)
8. Henschel, L., Conjeti, S., Estrada, S., Diers, K., Fischl, B., Reuter, M.: Fast Surfer—a fast and accurate deep learning-based neuroimaging pipeline. *NeuroImage*, 219, 117012 (2020)
9. Kavitha, C., Mani, V., Srividhya, S., Khalaf, O.I., Tavera Romero, C.A.: Early-stage Alzheimer's disease prediction using machine learning models. *Frontiers in Public Health*, 10, 853294 (2022)
10. Li, X., Chen, G., Shen, D.: Attention-aware convolutional neural network for Alzheimer's disease diagnosis. *IEEE Transactions on Medical Imaging*, 40, 2512–2521 (2021)
11. Sarraf, S., Tofighi, G.: DeepAD: Alzheimer's disease classification via deep convolutional neural networks using MRI and fMRI. *bioRxiv*, p. 070441 (2016)

12. Singh, A., Kumar, R.: Brain MRI image analysis for Alzheimer’s disease (AD) prediction using deep learning approaches. *SN Computer Science*, 5(1), 160 (2024)
13. Suk, H.I., Lee, S.W., Shen, D.: Deep learning-based feature representation for AD/MCI classification. *NeuroImage*, 101, 569–582 (2014)
14. Zhang, J., Wang, Y., Zu, C., Yu, B., Wang, L., Zhou, L.: Medical imaging-based diagnosis through machine learning and data analysis. In: *Advances in Artificial Intelligence, Computation, and Data Science: For Medicine and Life Science*, pp. 179–225 (2021)
15. Jack, C.R., et al.: NIA-AA Research Framework: Toward a biological definition of Alzheimer’s disease. *Alzheimer’s & Dementia*, 14(4), 535–562 (2018)
16. Dickerson, B.C., et al.: MRI-derived entorhinal and hippocampal atrophy in incipient and very mild Alzheimer’s disease. *Neurobiology of Aging*, 32(9), 1375–1382 (2011)
17. Rathore, S., Habes, M., Iftikhar, M.A., Shacklett, A., Davatzikos, C.: A review on neuroimaging-based classification studies and associated feature extraction methods for Alzheimer’s disease and its prodromal stages. *NeuroImage*, 155, 530–548 (2017)
18. Jo, T., Nho, K., Saykin, A.J.: Deep learning in Alzheimer’s disease: Diagnostic classification and prognostic prediction using neuroimaging data. *Frontiers in Aging Neuroscience*, 11, 220 (2019)
19. Weiner, M.W., et al.: The Alzheimer’s Disease Neuroimaging Initiative: a review of papers published since its inception. *Alzheimer’s & Dementia*, 11(6), e1–e120 (2015)
20. Liu, M., Cheng, D., Yan, W.: Classification of Alzheimer’s disease by combination of convolutional and recurrent neural networks using FDG-PET images. *Frontiers in Neuroinformatics*, 13, 63 (2021)

8. Annexure

01_Conversion of nii files to mgz file

```
import os
import subprocess
def convert_nii_to_mgz(input_dir, output_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for root, dirs, files in os.walk(input_dir):
        for file in files:
            if file.endswith(".nii"):
                input_file = os.path.join(root, file)
                output_file = os.path.join(output_dir, file.replace(".nii", ".mgz"))
                #Free Surfer command to convert input .nii file to output .mgz file
                command = f"mri_convert {input_file} {output_file}"
                try:
                    subprocess.run(command, shell=True, check=True)
                    print(f"Converted: {input_file} -> {output_file}")
                except subprocess.CalledProcessError as e:
                    print(f"Failed to convert {input_file}: {e}")

base_dir = "/home/soumya2000/MRI_Dataset"
output_base_dir = "/home/soumya2000/MRI_Dataset/Output"

folders = ["MCI", "CN", "AD"]

for folder in folders:
    input_dir = os.path.join(base_dir, folder)
    output_dir = os.path.join(output_base_dir, folder)
    convert_nii_to_mgz(input_dir, output_dir)
```

02_ROI Extraction

```
import os
import subprocess

input_base_dir = "Fast Surfer_Segmented_Files"
output_base_dir = "Label_final"
```

```
categories = ['AD', 'CN', 'MCI']

labels = '4 5 10 17 18 43 44 49 53 54 1006 2006'

for category in categories:
    input_category_dir = os.path.join(input_base_dir, category)
    output_category_dir = os.path.join(output_base_dir, category)

    for patient_id in os.listdir(input_category_dir):
        input_patient_dir = os.path.join(input_category_dir, patient_id)
        output_patient_dir = os.path.join(output_category_dir, patient_id)

        if not os.path.exists(output_patient_dir):
            os.makedirs(output_patient_dir)

        for file in os.listdir(input_patient_dir):
            if file.endswith('.mgz'):
                input_file = os.path.join(input_patient_dir, file)
                output_file = os.path.join(output_patient_dir, f"{patient_id}.mgz")

                command = f"mri_binarize --i {input_file} --match {labels} --o {output_file}"
                subprocess.run(command, shell=True)
```

03_Visualizing the output

```
from nilearn import plotting
from nilearn.image import load_img
import nibabel as nib
import matplotlib.pyplot as plt

path="Original_conformed_images/AD/ADNI_002_S_1018_MR_MPR____N3__Scaled_Br
_20070217032215330_S24312_I40828/ADNI_002_S_1018_MR_MPR____N3__Scaled_Br
_20070217032215330_S24312_I40828.mgz"
mri_image=nib.load(path)
image_data = mri_image.get_fdata()

# Display information about the image
print("Shape of MRI image:", image_data.shape)
print("Affine transformation matrix:\n", mri_image.affine)

# Visualize a specific slice (e.g., middle slice along the z-axis)
slice_idx = image_data.shape[2] // 2  # Middle slice
```

```
plt.imshow(image_data[:, :, slice_idx], cmap="gray")
plt.title(f"Slice {slice_idx}")
plt.axis("off")
plt.show()

#Visualizing the labels
path="Label_final/AD/ADNI_002_S_1018_MR_MPR____N3__Scaled_Br_2007021703221
5330_S24312_I40828/ADNI_002_S_1018_MR_MPR____N3__Scaled_Br_2007021703221
5330_S24312_I40828.mgz"
mri_image=nib.load(path)
image_data = mri_image.get_fdata()

# Display information about the image
print("Shape of MRI image:", image_data.shape)
print("Affine transformation matrix:\n", mri_image.affine)

# Visualize a specific slice (e.g., middle slice along the z-axis)
slice_idx = image_data.shape[2] // 2  # Middle slice
plt.imshow(image_data[:, :, slice_idx], cmap="gray")
plt.title(f"Slice {slice_idx}")
plt.axis("off")
plt.show()

#Merging the Raw MRI Image and the Fast Surfer Output Image
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt

def visualize_sagittal_slices(mgz_path, num_slices=64):
   # Load the MGZ image
   img = nib.load(mgz_path)
   img_data = img.get_fdata()
   # Compute slice intervals for sagittal (x-axis) slices
   x_slices = np.linspace(0, img_data.shape[0] - 1, num_slices, dtype=int)
   # Plot the sagittal slices
   cols = 8  # Number of slices per row
   rows = num_slices // cols
   fig, axes = plt.subplots(rows, cols, figsize=(20, rows * 3))
   axes = axes.ravel()
   fig.patch.set_facecolor('black')
   for i, x in enumerate(x_slices):
      axes[i].imshow(img_data[x, :, :], aspect='auto')
      axes[i].set_title(f"Sagittal {x}", color='white')  # Title in white
```

```
        axes[i].spines['top'].set_color('black')
        axes[i].spines['bottom'].set_color('black')
        axes[i].spines['left'].set_color('black')
        axes[i].spines['right'].set_color('black')
        axes[i].tick_params(axis='both', colors='black')  # Hide ticks
        axes[i].set_xticks([])
        axes[i].set_yticks([])
        axes[i].set_facecolor('black')  # Make sure the axis background is black too
    plt.tight_layout()
    plt.show()
mgz_path='/kaggle/input/labels-and-conformed-images/Labels/Labels/AD/ADNI_002_S_1018_MR_MPR____N3__Scaled_Br_20070217032215330_S24312_I40828/ADNI_002_S_1018_MR_MPR____N3__Scaled_Br_20070217032215330_S24312_I40828.mgz'
visualize_sagittal_slices(mgz_path, num_slices=64)

color_mgz_path='/kaggle/input/labels-and-conformed-images/Labels/Labels/CN/ADNI_002_S_0413_MR_MPR____N3__Scaled_2_Br_20081001114937668_S14782_I118675/ADNI_002_S_0413_MR_MPR____N3__Scaled_2_Br_20081001114937668_S14782_I118675.mgz'  # Fast Surfer output (hippocampus, amygdala, hypothalamus)
gray_nifti_path = '/kaggle/input/labels-and-conformed-images/Labels/Orig_CONFORMED/CN/ADNI_002_S_0413_MR_MPR____N3__Scaled_2_Br_20081001114937668_S14782_I118675/ADNI_002_S_0413_MR_MPR____N3__Scaled_2_Br_20081001114937668_S14782_I118675.mgz'  # Original NIfTI file

# Visualize sagittal slices (x-axis)
merge_and_visualize(color_mgz_path, gray_nifti_path, num_slices=24, axis='x')
# Visualize coronal slices (y-axis)
merge_and_visualize(color_mgz_path, gray_nifti_path, num_slices=24, axis='y')
# Visualize axial slices (z-axis)
merge_and_visualize(color_mgz_path, gray_nifti_path, num_slices=24, axis='z')

#see all MCI Slices
color_mgz_path = '/kaggle/input/labels-and-conformed-images/Labels/Labels/MCI/ADNI_002_S_0954_MR_MPR____N3__Scaled_2_Br_20081001120719118_S22322_I118694/ADNI_002_S_0954_MR_MPR____N3__Scaled_2_Br_20081001120719118_S22322_I118694.mgz'  # Fast Surfer output (hippocampus, amygdala, hypothalamus)
gray_nifti_path = '/kaggle/input/labels-and-conformed-images/Labels/Orig_CONFORMED/MCI/ADNI_002_S_0954_MR_MPR____N3__Scaled_2_Br_20081001120719118_S22322_I118694/ADNI_002_S_0954_MR_MPR____N3__Scaled_2_Br_20081001120719118_S22322_I118694.mgz'  # Original NIfTI file
```

```
# Visualize sagittal slices (x-axis)
merge_and_visualize(color_mgz_path, gray_nifti_path, num_slices=24, axis='x')
# Visualize coronal slices (y-axis)
merge_and_visualize(color_mgz_path, gray_nifti_path, num_slices=24, axis='y')
# Visualize axial slices (z-axis)
merge_and_visualize(color_mgz_path, gray_nifti_path, num_slices=24, axis='z')

# contouring
def contouring(image):

    # gaussian blur
    #gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # Applying threshold
    _, binary_image = cv2.threshold(blurred_image, 127, 255, cv2.THRESH_BINARY)

    # Ensure the binary image is of type CV_8UC1
    binary_image = binary_image.astype(np.uint8)

    # Detecting contours
    contours,_=cv2.findContours(binary_image,cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    # Drawing contours on the original image
    contour_image = image.copy()
    cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
    return contour_image

# Loading the .mgz file
mgz_file_path = '/kaggle/input/labels-and-conformed-
images/Labels/Orig_CONFORMED/AD/ADNI_002_S_1018_MR_MPR____N3__Scaled_B
r_20070217032215330_S24312_I40828/ADNI_002_S_1018_MR_MPR____N3__Scaled_B
r_20070217032215330_S24312_I40828.mgz'
mgz_data = nib.load(mgz_file_path)
mgz_file_path = '/kaggle/input/merged-image/AD_001.mgz'
mgz_data = nib.load(mgz_file_path)
# Get the image data as a NumPy array
image_data = mgz_data.get_fdata()

def hold_one_slice(image_data, axis=0, slice_index=0):
    if axis == 0:  # Coronal slice
        slice_data = image_data[slice_index, :, :]
    elif axis == 1:  # Sagittal slice
        slice_data = image_data[:, slice_index, :]
    else:  # Axial slice
        slice_data = image_data[:, :, slice_index]
    return slice_data
contoured_image=contouring(hold_one_slice(image_data,0,90))
show_slice(contoured_image,"example")
```

04_Building and Training the 3D-CNN and Other Model

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import nibabel as nib
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

def extract_roi_patches(subjects_df, roi_list, patch_size=(32, 32, 32)):

    roi_ids = {
        'left_hippocampus': 17,
        'right_hippocampus': 53,
        'left_amygdala': 18,
        'right_amygdala': 54,
        'left_hypothalamus': 801,
        'right_hypothalamus': 802,
        'left_entorhinal': 1006,
        'right_entorhinal': 2006
    }

    # Get ROI IDs to use
    roi_ids_to_use = {roi: roi_ids[roi] for roi in roi_list if roi in roi_ids}

    X_patches = []
    y = []
    subject_ids = []

    # Diagnostic mapping
    diagnosis_map = {'CN': 0, 'MCI': 1, 'AD': 2}

    for _, row in subjects_df.iterrows():
```

```
if row['diagnosis'] not in diagnosis_map:
    continue

# Load T1-weighted image
t1_path = os.path.join(os.path.dirname(row['segmentation']), 'orig.mgz')
if not os.path.exists(t1_path):
    continue

t1_img = nib.load(t1_path)
t1_data = t1_img.get_fdata()

# Load segmentation
seg_img = nib.load(row['segmentation'])
seg_data = seg_img.get_fdata()

# Process each ROI
for roi_name, roi_id in roi_ids_to_use.items():
    # Create mask
    roi_mask = (seg_data == roi_id)

    if np.sum(roi_mask) < 10:  # Skip if ROI is too small
        continue

    # Find center of mass
    coords = np.array(np.where(roi_mask)).mean(axis=1).astype(int)
    x, y, z = coords

    # Extract patch centered on ROI
    x_half, y_half, z_half = patch_size[0]//2, patch_size[1]//2, patch_size[2]//2

    # Handle boundary cases
    x_start = max(0, x - x_half)
    x_end = min(t1_data.shape[0], x + x_half)
    y_start = max(0, y - y_half)
    y_end = min(t1_data.shape[1], y + y_half)
    z_start = max(0, z - z_half)
    z_end = min(t1_data.shape[2], z + z_half)

    # Extract and pad if necessary
    patch = t1_data[x_start:x_end, y_start:y_end, z_start:z_end]

    # Pad if needed to ensure consistent size
    x_pad_before = max(0, x_half - x)
```

```
            x_pad_after = max(0, patch_size[0] - patch.shape[0] - x_pad_before)
            y_pad_before = max(0, y_half - y)
            y_pad_after = max(0, patch_size[1] - patch.shape[1] - y_pad_before)
            z_pad_before = max(0, z_half - z)
            z_pad_after = max(0, patch_size[2] - patch.shape[2] - z_pad_before)

            patch = np.pad(patch, ((x_pad_before, x_pad_after),
                                   (y_pad_before, y_pad_after),
                                   (z_pad_before, z_pad_after)), 'constant')

            # Normalize patch
            patch = (patch - np.min(patch)) / (np.max(patch) - np.min(patch) + 1e-8)

            # Add channel dimension
            patch = patch.reshape(*patch_size, 1)

            # Append to data
            X_patches.append(patch)
            y.append(diagnosis_map[row['diagnosis']])
            subject_ids.append(row['subject_id'])

    return np.array(X_patches), np.array(y), subject_ids

def build_3d_cnn(input_shape, num_classes=3):

    model = models.Sequential([
        # First convolutional block
        layers.Conv3D(32, kernel_size=3, activation='relu', padding='same',
input_shape=input_shape),
        layers.MaxPooling3D(pool_size=2),
        layers.BatchNormalization(),
        layers.Dropout(0.3),

        # Second convolutional block
        layers.Conv3D(64, kernel_size=3, activation='relu', padding='same'),
        layers.MaxPooling3D(pool_size=2),
        layers.BatchNormalization(),
        layers.Dropout(0.3),

        # Third convolutional block
        layers.Conv3D(128, kernel_size=3, activation='relu', padding='same'),
        layers.MaxPooling3D(pool_size=2),
        layers.BatchNormalization(),
        layers.Dropout(0.4),

        # Flatten and dense layers
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])

    # Compile model
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

def train_3d_cnn(X_patches, y, batch_size=8, epochs=50):
    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X_patches, y, test_size=0.2, random_state=42, stratify=y
    )

    # One-hot encode labels
    y_train_cat = to_categorical(y_train, num_classes=3)
    y_test_cat = to_categorical(y_test, num_classes=3)

    # Build model
    input_shape = X_train.shape[1:]
    model = build_3d_cnn(input_shape)

    # Defining callbacks
    callbacks = [
        tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True),
        tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=5)
    ]

    # Train model
    history = model.fit(
        X_train, y_train_cat,
        validation_data=(X_test, y_test_cat),
        epochs=epochs,
        batch_size=batch_size,
        callbacks=callbacks,
        class_weight={0: 1.0, 1: 1.0, 2: 1.0}  # Adjust if classes are imbalanced
    )

    # Evaluating model
    test_loss, test_acc = model.evaluate(X_test, y_test_cat)
    print(f"Test accuracy: {test_acc:.4f}")

    # Plot training history
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training')
    plt.plot(history.history['val_accuracy'], label='Validation')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training')
    plt.plot(history.history['val_loss'], label='Validation')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()

    # Generate predictions and confusion matrix
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    cm = confusion_matrix(y_test, y_pred_classes)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['CN', 'MCI', 'AD'],
yticklabels=['CN', 'MCI', 'AD'])
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix (3D CNN)')
    plt.show()
    return model, history
```

```
patch_size = (32, 32, 32)
X_patches, y_patches, subject_ids = extract_roi_patches(subjects_df, roi_list, patch_size)
cnn_model, cnn_history = train_3d_cnn(X_patches, y_patches)
```

05_Fast Surfer Installation and Generating Output

```
import os
import sys
from os.path import exists, join, basename, splitext

print("Starting setup. This could take a few minutes")
print("----------------------------------------------")

is_google_colab = "colab.research.google.com" in str(os.environ)
if is_google_colab:
    # this is for a Google Colab Notebook
    SETUP_DIR = "/content/"
else:
    # This is for Kaggle Notebook or local development
    SETUP_DIR = os.environ["HOME"] + "/Fast Surfer/"

# Go to the Fast Surfer directory
!mkdir -p "{SETUP_DIR}"
%cd "{SETUP_DIR}"

print(f"Using {SETUP_DIR} to store files.")

print("Downloading Fast Surfer")
print("----------------------------------------------")


git_repo_url = 'https://github.com/deep-mi/Fast Surfer.git'
project_name = splitext(basename(git_repo_url))[0]
FAST SURFER_HOME = SETUP_DIR + project_name + "/"
if not exists(project_name):
  # clone and install dependencies
  ! git clone -q --branch stable $git_repo_url
  ! pip install -r $FAST SURFER_HOME/requirements.txt
sys.path.append(FAST SURFER_HOME)

# Update dependencies
print("Installing required packages")
print("---------------------------------------------")

! pip install torchio==0.18.83
! pip install yacs==0.1.8
! pip install plotly==5.9.0

print("Finished setup")
print("---------------------------------------------")
```

5.1 Fast Surfer Segmentation

```
import os

input_dir = '/kaggle/input/alz-mgz-dataset/Output'
output_dir = '/kaggle/working/Fast Surfer_output'

os.makedirs(output_dir, exist_ok=True)
subdirs = ['AD','CN','MCI']

for subdir in subdirs:
   subdir_path = os.path.join(input_dir, subdir)
   output_subdir = os.path.join(output_dir, subdir)
   os.makedirs(output_subdir, exist_ok=True)

   for filename in os.listdir(subdir_path):
      if filename.endswith('.mgz'):
         subject_id = os.path.splitext(filename)[0]  # Remove the file extension
         t1_path = os.path.join(subdir_path, filename)
         output_file_path = os.path.join(output_subdir, subject_id, 'mri',
'aparc.DKTatlas+aseg.deep.mgz')

         if not os.path.exists(output_file_path):
            # Run Fast Surfer if the file hasn't been processed yet
            !FAST SURFER_HOME=$FAST SURFER_HOME \
             $FAST SURFER_HOME/run_Fast Surfer.sh --t1 "{t1_path}" \
                         --sd "{output_subdir}" \
                         --sid "{subject_id}" \
                         --seg_only \
                         --parallel --threads 4 \
                         --allow_root
            print(f"Processed: {subject_id}")
         else:
            print(f"Already processed: {subject_id}")
      else:
        print(f"Skipped non-mgz file: {filename}")

print("___________________________________")
print("Finished processing")
```

06_Statistical Approach for Improved Accuracy

```
BASE_DIR = "/kaggle/working/"
DATA_DIR = "/kaggle/input/Fast Surfer-stats/Fast Surfer_stats"
PROCESSED_DIR = os.path.join(BASE_DIR, "Processed_data")
MODELS_DIR = os.path.join(BASE_DIR, "Models")
RESULTS_DIR = os.path.join(BASE_DIR, "Results")
CM_DIR = os.path.join(RESULTS_DIR, "Confusion_Matrices")

os.makedirs(PROCESSED_DIR, exist_ok=True)
os.makedirs(MODELS_DIR, exist_ok=True)
os.makedirs(RESULTS_DIR, exist_ok=True)
os.makedirs(CM_DIR, exist_ok=True)

import glob
import matplotlib.pyplot as plt
import seaborn as sns
```

01: Extracting and Converting Stats Files to csv
1.1 For AD CLass

```
# Process AD class
ad_stats_files = glob.glob(os.path.join(DATA_DIR, "AD", "*.stats"))
ad_data = []

print(f"Processing {len(ad_stats_files)} files in AD directory...")
for file_path in ad_stats_files:
  try:
    data = {}
    subject_id = os.path.basename(file_path).replace('.stats', '')
    data['subject_id'] = subject_id
    data['label'] = "AD"

    with open(file_path, 'r') as f:
      lines = f.readlines()

    data_start = 0
```

```
        for i, line in enumerate(lines):
            if line.startswith('# ColHeaders'):
                data_start = i + 1
                break

        for line in lines[data_start:]:
            if line.strip() and not line.startswith('#'):
                parts = line.strip().split()
                if len(parts) >= 5:
                    index = int(parts[0])
                    struct_name = parts[4].replace('-', '_')

                    if len(parts) >= 4:
                        volume = float(parts[3])
                        data[f"{struct_name}_volume"] = volume

                    if len(parts) >= 6:
                        norm_mean = float(parts[5])
                        data[f"{struct_name}_mean"] = norm_mean

                    if len(parts) >= 7:
                        norm_std = float(parts[6])
                        data[f"{struct_name}_std"] = norm_std

        ad_data.append(data)
        print("Processed",subject_id)
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
print("All AD data has been Processed")
```

1.2 For CN Class

```
cn_stats_files = glob.glob(os.path.join(DATA_DIR, "CN", "*.stats"))
cn_data = []

print(f"Processing {len(cn_stats_files)} files in CN directory...")
for file_path in cn_stats_files:
    try:
        data = {}
        subject_id = os.path.basename(file_path).replace('.stats', '')
        data['subject_id'] = subject_id
        data['label'] = "CN"

        with open(file_path, 'r') as f:
            lines = f.readlines()
        data_start = 0
        for i, line in enumerate(lines):
            if line.startswith('# ColHeaders'):
                data_start = i + 1
                break

        for line in lines[data_start:]:
            if line.strip() and not line.startswith('#'):
                parts = line.strip().split()
                if len(parts) >= 5:
                    index = int(parts[0])
                    struct_name = parts[4].replace('-', '_')

                    if len(parts) >= 4:
                        volume = float(parts[3])
                        data[f"{struct_name}_volume"] = volume

                    if len(parts) >= 6:
                        norm_mean = float(parts[5])
                        data[f"{struct_name}_mean"] = norm_mean

                    if len(parts) >= 7:
                        norm_std = float(parts[6])
                        data[f"{struct_name}_std"] = norm_std

        cn_data.append(data)
        print("Processed",subject_id)
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
print("All CN data has been Processed")
```

1.3 For MCI Class

In [6]:

```
mci_stats_files = glob.glob(os.path.join(DATA_DIR, "MCI", "*.stats"))
mci_data = []

print(f"Processing {len(mci_stats_files)} files in MCI directory...")
for file_path in mci_stats_files:
    try:
        data = {}
        subject_id = os.path.basename(file_path).replace('.stats', '')
        data['subject_id'] = subject_id
        data['label'] = "MCI"

        with open(file_path, 'r') as f:
            lines = f.readlines()

        # Find where the data starts
        data_start = 0
        for i, line in enumerate(lines):
            if line.startswith('# ColHeaders'):
                data_start = i + 1
                break

        # Extract the data
        for line in lines[data_start:]:
            if line.strip() and not line.startswith('#'):
                parts = line.strip().split()
                if len(parts) >= 5:
                    index = int(parts[0])
                    struct_name = parts[4].replace('-', '_')

                    if len(parts) >= 4:
                        volume = float(parts[3])
                        data[f"{struct_name}_volume"] = volume

                    if len(parts) >= 6:
                        norm_mean = float(parts[5])
                        data[f"{struct_name}_mean"] = norm_mean

                    if len(parts) >= 7:
                        norm_std = float(parts[6])
                        data[f"{struct_name}_std"] = norm_std

        mci_data.append(data)
        print("Processed",subject_id)
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
print("All MCI data has been Processed")
```

02: Combining All CSVs and Labelling Data

```
ad_df = pd.DataFrame(ad_data)
cn_df = pd.DataFrame(cn_data)
mci_df = pd.DataFrame(mci_data)
```

```
# Combining all data
combined_data = pd.concat([ad_df, cn_df, mci_df], ignore_index=True)

# Fill missing values with 0
combined_data = combined_data.fillna(0)

# Save the combined raw data
combined_data.to_csv(os.path.join(PROCESSED_DIR, "combined_raw.csv"), index=False)

print(f"Combined data shape: {combined_data.shape}")
print(f"Class distribution: {combined_data['label'].value_counts()}")
```

03: Splitting the data

```
In [8]:
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
In [9]:
# Making a copy to avoid modifying the original DataFrame
df = combined_data.copy()
df.head(10)

# Separate features and target
X = df.drop(['label', 'subject_id'], axis=1, errors='ignore')
y = df['label']

# Keep track of feature names for later use
feature_names = X.columns.tolist()

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into train, validation, and test sets (70%, 15%, 15%)
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X_scaled, y, test_size=0.15, random_state=42, stratify=y
)

X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.1765, random_state=42, stratify=y_train_val
)  # 0.1765 * 0.85 = 0.15

# Create DataFrames for saving to CSV
```

```
train_df = pd.DataFrame(X_train, columns=feature_names)
train_df['label'] = y_train.values

val_df = pd.DataFrame(X_val, columns=feature_names)
val_df['label'] = y_val.values

test_df = pd.DataFrame(X_test, columns=feature_names)
test_df['label'] = y_test.values

# Save to CSV
train_df.to_csv(os.path.join(PROCESSED_DIR, "train.csv"), index=False)
val_df.to_csv(os.path.join(PROCESSED_DIR, "val.csv"), index=False)
test_df.to_csv(os.path.join(PROCESSED_DIR, "test.csv"), index=False)

print("Data split and saved:")
print(f"Train set: {X_train.shape[0]} samples")
print(f"Validation set: {X_val.shape[0]} samples")
print(f"Test set: {X_test.shape[0]} samples")
```

04: Training Model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Reshape, Conv2D, MaxPooling2D,
Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
import joblib
import warnings
warnings.filterwarnings('ignore')
```

4.1 Label Encoding & One hot Encoding

```
In [13]:
## Convert string labels to numerical
label_encoder = LabelEncoder()
y_train_numeric = label_encoder.fit_transform(y_train)
y_val_numeric = label_encoder.transform(y_val)
y_test_numeric = label_encoder.transform(y_test)

# Get input shape and number of classes
input_size = X_train.shape[1]
num_classes = len(label_encoder.classes_)
```

```
# Convert to one-hot encoding for Keras
y_train_onehot = tf.keras.utils.to_categorical(y_train_numeric, num_classes=num_classes)
y_val_onehot = tf.keras.utils.to_categorical(y_val_numeric, num_classes=num_classes)
y_test_onehot = tf.keras.utils.to_categorical(y_test_numeric, num_classes=num_classes)

# Calculate reshape dimensions
reshape_size = int(np.sqrt(input_size)) + 1
pad_size = (reshape_size**2) - input_size
```
4.2 Reshaping Data for 2D CNN
In [14]:
```
X_train_padded = np.pad(X_train, ((0, 0), (0, pad_size)), mode='constant')
X_val_padded = np.pad(X_val, ((0, 0), (0, pad_size)), mode='constant')
X_test_padded = np.pad(X_test, ((0, 0), (0, pad_size)), mode='constant')

X_train_reshaped = X_train_padded.reshape(-1, reshape_size, reshape_size, 1)
X_val_reshaped = X_val_padded.reshape(-1, reshape_size, reshape_size, 1)
X_test_reshaped = X_test_padded.reshape(-1, reshape_size, reshape_size, 1)

print("Size of the Reshaped Train data:",X_train_reshaped.shape)
print("Size of the Reshaped Validation data",X_val_reshaped.shape)
print("Size of the Reshaped Test data",X_test_reshaped.shape)
```

4.3 Defining the Model
```
model = Sequential([
  Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(reshape_size, reshape_size, 1)),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(128, activation='relu'),
  Dropout(0.3),
  Dense(64, activation='relu'),
  Dense(num_classes, activation='softmax')
])
model.summary()
```

4.4 Compiling and training the model
In [16]:
```
# Compile the model
```

```
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
In [17]:
# Define callbacks
early_stopping = EarlyStopping(monitor='val_accuracy', patience=30,
restore_best_weights=True)
model_checkpoint = ModelCheckpoint(
    os.path.join(MODELS_DIR, 'best_cnn_model.h5'),
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1
)

# Train the model
history = model.fit(
    X_train_reshaped,
    y_train_onehot,
    validation_data=(X_val_reshaped, y_val_onehot),
    epochs=100,
    batch_size=32,
    callbacks=[early_stopping, model_checkpoint],
    verbose=1
)
```

4.5 Vizualizing the CNN Model

```
In [18]:
# Save the label encoder
joblib.dump(label_encoder, os.path.join(MODELS_DIR, 'label_encoder.joblib'))

# Evaluate the model on validation set
y_val_pred_proba = model.predict(X_val_reshaped)
y_val_pred_classes = np.argmax(y_val_pred_proba, axis=1)
y_val_pred_labels = label_encoder.inverse_transform(y_val_pred_classes)

# Display CNN performance metrics
cnn_accuracy = accuracy_score(y_val, y_val_pred_labels)
cnn_precision = precision_score(y_val, y_val_pred_labels, average='weighted')
cnn_recall = recall_score(y_val, y_val_pred_labels, average='weighted')
cnn_f1 = f1_score(y_val, y_val_pred_labels, average='weighted')
```

```
print("\nCNN Model Performance:")
print(f"Validation Accuracy: {cnn_accuracy:.4f}")
print(f"Validation Precision: {cnn_precision:.4f}")
print(f"Validation Recall: {cnn_recall:.4f}")
print(f"Validation F1 Score: {cnn_f1:.4f}")

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.savefig(os.path.join(RESULTS_DIR, 'cnn_training_history.png'))
plt.show()
# Create CNN confusion matrix
cm = confusion_matrix(y_val, y_val_pred_labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
        xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - CNN')
plt.tight_layout()
plt.savefig(os.path.join(CM_DIR, 'CNN_cm.png'))
plt.show()
```

05: Training Other Models(RandomForest, SVM, KNN, XGB)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

```
import xgboost as xgb
```

5.1 Models

In [22]:

```
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'XGBoost': xgb.XGBClassifier(n_estimators=100, random_state=42, use_label_encoder=False, eval_metric='mlogloss'),
    'SVM': SVC(probability=True, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42)
}

results = []
val_predictions = {}
```

5.2 Training and Evaluating the Models

```
for name, model_instance in models.items():
    print(f"Training {name}......")

    try:
        # Train the model
        model_instance.fit(X_train, y_train_numeric)

        # Make predictions
        y_val_pred_enc = model_instance.predict(X_val)

        # Convert encoded predictions back to original labels
        y_val_pred = label_encoder.inverse_transform(y_val_pred_enc)

        val_predictions[name] = y_val_pred

        # Compute metrics
        accuracy = accuracy_score(y_val, y_val_pred)
        precision = precision_score(y_val, y_val_pred, average='weighted')
        recall = recall_score(y_val, y_val_pred, average='weighted')
        f1 = f1_score(y_val, y_val_pred, average='weighted')

        results.append({
            'Model': name,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1 Score': f1
        })

        # Save model
        model_path = os.path.join(MODELS_DIR, f"{name.replace(' ', '_')}.joblib")
        joblib.dump(model_instance, model_path)
        print(f"  Saved model to {model_path}")

    except Exception as e:
        print(f"  Error training {name}: {e}")

# Save Other model metrics to CSV
trad_results_df = pd.DataFrame(results)
metrics_path = os.path.join(RESULTS_DIR, 'traditional_models_metrics.csv')
trad_results_df.to_csv(metrics_path, index=False)
print(f"Saved Other model metrics to {metrics_path}")
```

5.3 Confusion Matrices

```
for name, y_pred in val_predictions.items():
    print(f"Creating confusion matrix for {name}...")

    # Create and save confusion matrix
    cm = confusion_matrix(y_val, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title(f'Confusion Matrix - {name}')
    plt.tight_layout()
    cm_path = os.path.join(CM_DIR, f"{name.replace(' ', '_')}_cm.png")
    plt.savefig(cm_path)
    plt.show()
    print(f"  Saved confusion matrix to {cm_path}")
```

5.4 Feature Importance

```
if 'Random Forest' in models:
    print("Creating feature importance plot...")
    feature_importance = pd.DataFrame({
        'Feature': feature_names,
        'Importance': models['Random Forest'].feature_importances_
    })
    feature_importance = feature_importance.sort_values('Importance',
ascending=False).head(20)
```

```
    plt.figure(figsize=(10, 8))
    sns.barplot(x='Importance', y='Feature', data=feature_importance)
    plt.title('Top 20 Feature Importances (Random Forest)')
    plt.tight_layout()
    importance_path = os.path.join(RESULTS_DIR, 'feature_importance.png')
    plt.savefig(importance_path)
    plt.show()
    print(f"Saved feature importance plot to {importance_path}")
Creating feature importance plot...
In [26]:
cnn_results = pd.DataFrame([{
    'Model': 'CNN',
    'Accuracy': cnn_accuracy,
    'Precision': cnn_precision,
    'Recall': cnn_recall,
    'F1 Score': cnn_f1
}])

# Combine all results
all_results_df = pd.concat([trad_results_df, cnn_results], ignore_index=True)
combined_metrics_path = os.path.join(RESULTS_DIR, 'all_models_metrics.csv')
all_results_df.to_csv(combined_metrics_path, index=False)
print(f"Saved combined metrics to {combined_metrics_path}")
```

5.5 Creating individual metric comparison plots

```
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']

for metric in metrics:
    plt.figure(figsize=(12, 6))
    sns.barplot(x='Model', y=metric, data=all_results_df)
    plt.title(f'{metric} Comparison')
    plt.xticks(rotation=45)
    plt.tight_layout()
    metric_path = os.path.join(RESULTS_DIR, f"{metric.lower().replace(' ', '_')}_comparison.png")
    plt.savefig(metric_path)
    plt.show()
    print(f"Saved {metric} comparison plot to {metric_path}")
```

5.6 Creating Combined Metrics Plot

```
plt.figure(figsize=(14, 8))
```

```
results_melted = pd.melt(all_results_df, id_vars=['Model'], value_vars=metrics,
                var_name='Metric', value_name='Score')
sns.barplot(x='Model', y='Score', hue='Metric', data=results_melted)
plt.title('Model Performance Comparison')
plt.xticks(rotation=45)
plt.tight_layout()
comparison_path = os.path.join(RESULTS_DIR, 'model_performance_comparison.png')
plt.savefig(comparison_path)
plt.show()
print(f"Saved overall performance comparison plot to {comparison_path}")
```

06: Finding Best Model

In [29]:

```
best_row = all_results_df.loc[all_results_df['F1 Score'].idxmax()]
best_model_name = best_row['Model']
best_f1 = best_row['F1 Score']

print(f"Best model: {best_model_name} with F1 Score: {best_f1:.4f}")

# Evaluate the best model on test set
if best_model_name == 'CNN':
   # Load best CNN model
   best_model            =            tf.keras.models.load_model(os.path.join(MODELS_DIR,
'best_cnn_model.h5'))

   # Make predictions on test set
   y_test_pred_proba = best_model.predict(X_test_reshaped)
   y_test_pred_classes = np.argmax(y_test_pred_proba, axis=1)
   y_test_pred_labels = label_encoder.inverse_transform(y_test_pred_classes)

   # Save the model with additional metadata
   model_info = {
      'name': 'CNN',
      'scaler': scaler,
      'label_encoder': label_encoder,
   }
   joblib.dump(model_info, os.path.join(MODELS_DIR, 'best_model_info.joblib'))

else:
   # Load Other models
   best_model  =  joblib.load(os.path.join(MODELS_DIR,  f"{best_model_name.replace('  ',
'_')}.joblib"))

   # Make predictions on test set
```

```
    y_test_pred_enc = best_model.predict(X_test)
    y_test_pred_labels = label_encoder.inverse_transform(y_test_pred_enc)

    # Save as best model with metadata
    joblib.dump({
        'model': best_model,
        'scaler': scaler,
        'label_encoder': label_encoder,
        'name': best_model_name
    }, os.path.join(MODELS_DIR, 'best_model.joblib'))
```

07: Evaluating Final Test Metrics

```
test_accuracy = accuracy_score(y_test, y_test_pred_labels)
test_precision = precision_score(y_test, y_test_pred_labels, average='weighted')
test_recall = recall_score(y_test, y_test_pred_labels, average='weighted')
test_f1 = f1_score(y_test, y_test_pred_labels, average='weighted')

print("\nFinal Test Results:")
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Precision: {test_precision:.4f}")
print(f"Test Recall: {test_recall:.4f}")
print(f"Test F1 Score: {test_f1:.4f}")

test_report = classification_report(y_test, y_test_pred_labels, output_dict=True)
test_metrics_df = pd.DataFrame(test_report).transpose()
test_metrics_df.to_csv(os.path.join(RESULTS_DIR, 'test_metrics.csv'))

cm = confusion_matrix(y_test, y_test_pred_labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
        xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Test Set')
plt.tight_layout()
plt.savefig(os.path.join(RESULTS_DIR, 'test_confusion_matrix.png'))
plt.show()

print("\nAlzheimer's Disease Detection Pipeline has been completed successfully!")
print(f"Results saved to {RESULTS_DIR}")
print(f"Models saved to {MODELS_DIR}")
```

DrillBit

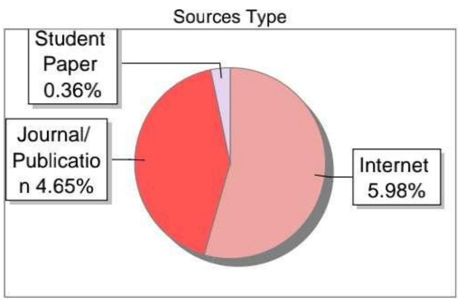
The Report is Generated by DrillBit Plagiarism Detection Software

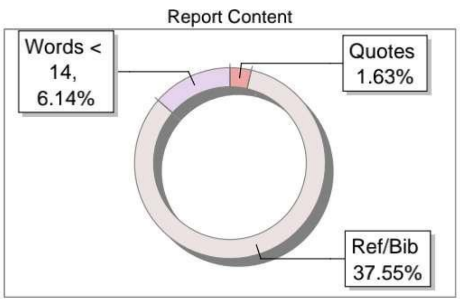
Submission Information

Author Name	Anubhab Halder
Title	IDENTIFICATION OF ALZHEIMER'S DISEASE FROM MRI IMAGES USING MACHINE LEARNING TOOLS
Paper/Submission ID	3660936
Submitted by	aiml@heritageit.edu
Submission Date	2025-05-24 15:02:09
Total Pages, Total Words	58, 9888
Document type	Project Work

Result Information

Similarity **11 %**





Exclude Information

Quotes	Not Excluded
References/Bibliography	Not Excluded
Source: Excluded < 14 Words	Not Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File