

Alzheimer's Disease Detection Using Machine Learning

A Project Report

Submitted in partial fulfillment of the requirements for the award of
degree of

Computer Science and Engineering

(Data Science and Machine Learning)

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB



From 02/06/24 to 05/02/24

SUBMITTED BY

Name of Student

Ritu Priya Singh (12001595)
Saloni Singh (12012579)
Fateh Singh Rana (12010320)
Dikshit Kukreja (12009356)
Akarsh Anand (12010997)
Neelesh Sharma (12010517)

Name of the Supervisor

Sandeep Kumar

UID of Supervisor

28953

Alzheimer's Disease Detection Using Machine Learning

A Project Report

Submitted in partial fulfillment of the requirements for the award of
degree of

Computer Science and Engineering
(Data Science and Machine Learning)

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB



From 02/06/24 to 05/02/24

SUBMITTED BY

Name of Student

Ritu Priya Singh (12001595)

Saloni Singh (12012579)

Fateh Singh Rana (12010320)

Dikshit Kukreja (12009356)

Akarsh Anand (12010997)

Neelesh Sharma (12010517)

Ritu Priya Singh

Saloni

Fateh

Dikshit

Akarsh

Neelesh

Signature of Student

Name of the Supervisor

Sandeep Kumar

UID of Supervisor

28953

Sandeep Kumar

Signature of Supervisor

**TOPIC APPROVAL PERFORMANCE**

School of Computer Science and Engineering (SCSE)

Program : P132::B.Tech. (Computer Science and Engineering)

COURSE CODE : CSE445

REGULAR/BACKLOG : Regular

GROUP NUMBER : CSERGC0135

Supervisor Name : Sandeep Kumar

UID : 28953

Designation : Assistant Professor

Qualification : _____

Research Experience : _____

SR.NO.	NAME OF STUDENT	Prov. Regd. No.	BATCH	SECTION	CONTACT NUMBER
1	Saloni Singh	12012579	2020	K20DW	7060712807
2	Akarsh Anand	12010997	2020	K20ND	9354281762
3	Neelesh Sharma	12010517	2020	K20MK	8627024789
4	Fateh Singh Rana	12010320	2020	K20DP	8264132918
5	Dikshit Kukreja	12009356	2020	K20HS	9215823555
6	Ritu Priya Singh	12001595	2020	K20MR	9910901975

SPECIALIZATION AREA : Database Systems-I

Supervisor Signature: _____

PROPOSED TOPIC : Alzheimer's Disease Detection Using Machine Learning

Qualitative Assessment of Proposed Topic by PAC		
Sr.No.	Parameter	Rating (out of 10)
1	Project Novelty: Potential of the project to create new knowledge	7.10
2	Project Feasibility: Project can be timely carried out in-house with low-cost and available resources in the University by the students.	7.00
3	Project Academic Inputs: Project topic is relevant and makes extensive use of academic inputs in UG program and serves as a culminating effort for core study area of the degree program.	7.00
4	Project Supervision: Project supervisor's is technically competent to guide students, resolve any issues, and impart necessary skills.	7.60
5	Social Applicability: Project work intends to solve a practical problem.	7.30
6	Future Scope: Project has potential to become basis of future research work, publication or patent.	7.00

PAC Committee Members		
PAC Member (HOD/Chairperson) Name: Dr. Virrat Devaser	UID: 14591	Recommended (Y/N): Yes
PAC Member (Allied) Name: Dr. Shilpa Sharma	UID: 13891	Recommended (Y/N): Yes
PAC Member 3 Name: Dr. Amritpal Singh	UID: 17673	Recommended (Y/N): Yes

Final Topic Approved by PAC: Alzheimer's Disease Detection Using Machine Learning

Overall Remarks: Approved

PAC CHAIRPERSON Name: 17442::Dr. Arun Malik

Approval Date: 08 Apr 2024

4/30/2024 12:53:40 PM

Declaration by Student

To whom so ever it may concern

I, Ritu Priya Singh (12001595), Saloni Singh (12012579), Fateh Singh Rana (12010320), Dikshit Kukreja (12009356), Akarsh Anand (12010997), Neelesh Sharma (12010517), hereby declare that the work done by me on “Alzheimer's Disease Detection Using Machine Learning” under the supervision of Sandeep Kumar, Assistant Professor, Lovely professional University, Phagwara, Punjab, is a record of original work for the partial fulfilment of the requirements for the award of the degree, Bachelor of Technology in Computer Science and Engineering.

Ritu Priya Singh (12001595)

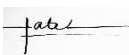
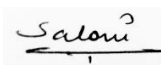
Saloni Singh (12012579)

Fateh Singh Rana (12010320)

Dikshit Kukreja (12009356)

Akarsh Anand (12010997)

Neelesh Sharma (12010517)



Signature of the student

Dated:

Declaration by the Supervisor

To whom so ever it may concern

This is to certify that Ritu Priya Singh (12001595), Saloni Singh (12012579), Fateh Singh Rana (12010320), Dikshit Kukreja (12009356), Akarsh Anand (12010997), Neelesh Sharma (12010517), from Lovely Professional University, Phagwara, Punjab, has worked on “Alzheimer's Disease Detection Using Machine Learning” under my supervision from. It is further stated that the work carried out by the student is a record of original work to the best of my knowledge for the partial fulfilment of the requirements for the award of the degree, Bachelor of Technology in Computer Science and Engineering.

Sandeep Kumar

Name of Supervisor

28953

UID of Supervisor

Signature of Supervisor

Acknowledgement

We would like to express our deepest gratitude to all those who contributed to the completion of this capstone project. Firstly, we extend our sincere appreciation to Mr. Sandeep Kumar (28953), for their invaluable guidance, encouragement, and unwavering support throughout the research process. Their expertise and insights have been instrumental in shaping the direction and quality of this work.

We are also grateful to the members of our team, Ritu Priya Singh (12001595), Fateh Singh Rana (12010320), Akarsh Anand (12010997), Dikshit Kukreja (12009356), Saloni Singh (12012579), Neelesh Sharma (12010517), whose collaborative efforts and dedication greatly enriched the scope and depth of this study. Their contributions in data collection, analysis, and discussion were indispensable to the project's success.

Furthermore, we would like to acknowledge the assistance provided by Lovely Professional University in facilitating access to resources and facilities necessary for conducting this research. The logistical support and academic environment provided by the institution have been crucial in fostering our academic pursuits.

This project would not have been possible without the generosity, guidance, and support of all those mentioned above. Thank you for being part of this research.

List of Contents

S. No.	Title	Page
1	Declaration by Student	i
2	Declaration by Supervisor	ii
3	Acknowledgement	iii
4	Abstract	v
5	List of Tables	vi
6	List of Figures	vii
7	List of Abbreviations	x
8	Chapter-1 Introduction	1
9	Chapter-2 Review of Literature	5
10	Chapter-3 Implementation of Project	20
11	Chapter-4 Results and Discussions	26
12	Final Chapter- Conclusion	56
13	Publication Details	58
14	References	59
15	Plagiarism Report	62

Abstract

The study investigates the application of advanced machine learning methods, specifically Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), in the field of Alzheimer's disease (AD) research. AD is a progressive neurological disorder characterized by memory loss and cognitive decline, affecting a significant number of individuals worldwide, particularly the elderly population. The.

By using CNNs and RNNs, which are known for analyzing spatial and sequential data respectively, the research aims to gain a deeper understanding of the progression of Alzheimer Disease. RNNs are particularly useful in tracking changes over time, while CNNs is used in identifying patterns in brain scans, such as MRI images, which play a crucial role in AD research.

The study utilizes widely available datasets from reputable sources like OASIS and Kaggle, which are known for their collection of publicly available brain imaging data. The results demonstrate promising performance, with CNNs achieving an accuracy rate of 100% on the OASIS dataset and 97.33% on Kaggle. Similarly, RNNs achieve accuracy rates of 73.15% and 80.67% on the same datasets.

These findings show the potential of machine learning in detection of Alzheimer Disease, as well as enhancing the process of its diagnosis and treatment of the disease. They show the significance of using diverse datasets and Machine Learnings techniques to tackle the problems associated with AD research and to develop effective solution for this complex condition.

List of Tables:

S. No.	Title	Page No.
1	List of Figures	vii
2	Literature Review Summary	16

List of Figures

Figure No.	Caption	Page No.
1	Code snippet for Separable Convolutional Block with Batch Normalization & MaxPool2D	22
2	Dense Block with Dropout and Batch Normalization	23
3	Sequential RNN and Dense Layers Setup	23
4	Model Architecture of our Project	25
5	Importing necessary libraries and TPU Configuration for our model implementation	27
6	Deep Learning Hyperparameters	27
7	Image Data Generators Setup	28
8	Fetching Batch of Images and Labels	29
9	Concatenate Train and Test Data	29
10	Split Data into Train, Validate, Test Sets	30
11	Data and Labels Shapes Output	30
12	Displaying Sample Images from Train, Validation, and Test Sets	30
13	Displaying Sample Images from Train, Validation, and Test Sets	31
14	Displaying Maximum and Minimum Pixel Values	31
15	Displaying Separate Channels and Combined Image	32
16	Function of a Convolutional Block	33
17	Defining of a Dense Block	33
18	Model Building Function	34
19	Model Compilation and Metric Definitions	35
20	Learning Rate Scheduler and Early Stopping Callbacks	36

21	Model Training with Training and Validation Data	36
22	Making Predictions on Test Data	37
23	Predicted Labels	37
24	Evaluation of CNN Model Accuracy	37
25	Summary of the CNN Model Architecture	38
26	Visualization of Training and Validation Metrics	39
27	Classification Report for Model Evaluation	39
28	Plotting Training and Validation Accuracy	40
29	Reshaping Data for RNN	40
30	Building an RNN Model	41
31	Training and Evaluating RNN Model	42
32	Plotting Training and Validation Accuracy for RNN Model	43
33	Importing Necessary Libraries	44
34	Gathering Image File Paths for Different Classes	44
35	Displaying Sample Images from Each Class	44
36	Images from Each Class	45
37	Printing the Number of Images in Each Class	45
38	Trimming the Number of Images in Each Class to Balance the Dataset	45
39	Printing the Updated Number of Images in Each Class	45
40	Performing One Hot Encoding	46
41	Processing Images and Labels for Model Training	46
42	Converting Data to NumPy Array	47
43	Shape of the Data Array	47
44	Converting Labels to NumPy Array	47
45	Splitting Data into Training and Testing Sets	47

46	Creating a CNN Model	48
47	Summary of the CNN Model Architecture	49
48	Compiling the CNN Model	49
49	Setting up Early Stopping Callback	50
50	Training the CNN Model with Early Stopping Callback	50
51	Evaluating the CNN Model Accuracy	50
52	Plotting Training and Validation Accuracy for CNN Model	51
53	Plotting Training and Validation Loss for CNN Model	51
54	Importing LSTM Layer	52
55	Removing the Last Layer of the Model	52
56	Setting up Input Shape for the LSTM Layer	52
57	Adding LSTM Layers to the Model	52
58	Compiling the RNN Model	52
59	Training the RNN Model with Early Stopping Callback	52
60	Evaluating the RNN Model Accuracy	53
61	Plotting Training and Validation Accuracy for RNN Model	53
62	Plotting Training and Validation Loss for RNN Model	54
63	Performance comparison of proposed model with existing models	55

List of Abbreviations:

- CNN – Convolutional Neural Network
- RNN - Recurrent Neural Networks
- LSTM – Long Short-Term Memory
- WHO – World Health Organization
- OASIS – Outcome and Assessment Information Set
- AD - Alzheimer’s Disease
- MRI – Magnetic Resonance Imaging
- TPU – Tensor Processing Unit
- EEG- Electroencephalography
- IMV – Iterative Model Validation
- API – Application Package Installer
- RGB – Red, Green, Blue
- PIL – Python Imaging Libraries
- AUC – Area Under Curve
- NLP – Natural Language Processing
- ADNI - Alzheimer's Disease Neuroimaging Initiative
- AIBL - Australian Imaging, Biomarkers & Lifestyle
- MIRIAD - Minimal Interval Resonance Imaging
- MLP – Multilayer Perceptron
- SVM – Support Vector Machine
- ANN – Artificial Neural Network
- SMO – Sequential Minimal Optimization
- PET - Positron Emission Tomography
- SNUBH – Seoul National University Bundang Hospital
- CN-EMCI-LMCI-AD - Cognitively Normal-Early Mild Cognitive Impairment-Late Mild Cognitive Impairment-Alzheimer's Disease

Chapter 1: Introduction

A tremendous demand on people, families, and healthcare systems, Alzheimer's disease affects millions of people globally and is one of the most urgent health issues of our day. The incidence of Alzheimer's disease is rising worldwide as the population ages, underscoring the critical need for novel strategies for the early diagnosis, prognosis, and treatment of this crippling illness. The majority of those affected by Alzheimer's disease are older adults, and the risk rises with age. Although the precise origin of Alzheimer's disease is unknown, a complex interaction of genetic, environmental, and lifestyle factors is thought to be involved. It is anticipated that the prevalence of Alzheimer's disease will increase dramatically as the population ages, presenting serious problems to society and healthcare systems.

Because Alzheimer's is a complex illness with mild early signs, early detection is essential for appropriate interventions and individualized treatment programs. Nevertheless, precise diagnosis of the disease is still challenging. Neuroimaging data can be used to uncover patterns suggestive of Alzheimer's disease, and machine learning methods hold promise for improving early identification and prognosis of the condition. Alzheimer's patients benefit greatly from prompt interventions, better patient outcomes, and less demand on healthcare resources and caregivers. Supervised, unsupervised, and deep learning are a few examples of machine learning approaches that provide strong tools for deciphering complicated datasets and identifying patterns associated with Alzheimer's disease. Classification of brain images is important for the diagnosis and management of neurological conditions, including Alzheimer's disease. Classification techniques' efficacy and precision, however, differ according to the model's architecture and the dataset. This paper uses Convolutional Neural Networks and Recurrent Neural Networks to accurately classify brain pictures into numerous categories. The objective is to examine these models' performance on two different datasets that were acquired from the Kaggle and OASIS communities. A major issue in medical imaging analysis is accurately

classifying brain images into different categories, especially for neurological illnesses like Alzheimer's disease. Although both architectures are good at classification, CNNs are better at capturing spatial information while RNNs are better at comprehending temporal linkages and sequential patterns. Still, differences in accuracy rates among the topologies emphasize how crucial it is to choose the right model depending on the specifics of the task and the dataset.

Scope Of This Project:

The project's scope encompasses the following key components:

- **Data Gathering:** Collecting a variety of datasets containing biomarkers, clinical measures, demographic details, and other pertinent variables related to Alzheimer's disease.
- **Feature Identification:** Selecting informative features and variables from the amassed data to serve as inputs for predictive modeling.
- **Model Creation:** Developing machine learning algorithms capable of discerning patterns indicative of Alzheimer's disease onset and progression.
- **Model Verification:** Evaluating the performance of predictive models using independent datasets and validation methods to ensure their reliability and applicability.
- **Model Implementation:** Integrating predictive models into clinical practice, healthcare decision-making, and public health initiatives to aid in Alzheimer's disease prevention and management.
- **Integration of Multimodal Data:** Investigating how different data types, including genetic, imaging, clinical, and behavioral data, might be combined to improve the precision and robustness of predictive models.
- **Continuous Assessment:** Assessing predictive model performance on a regular basis in real-world scenarios and adjusting based on input from stakeholders, patients, and healthcare practitioners.
- **Enhancing Early Detection:** Creating techniques to identify Alzheimer's early on when treatment is most effective.
- **Treatment Optimization:** Enhance patient outcomes by optimizing treatment approaches by the prediction of disease start and progression.

- Reducing burden: Using predictive models to support proactive management techniques that benefit caregivers and healthcare systems.
- Research Advancement: Finding novel biomarkers and treatment targets for Alzheimer's disease by utilizing machine learning techniques.
- Research Advancement: Finding novel biomarkers and treatment targets for Alzheimer's disease by utilizing machine learning techniques.
- Improving Quality of Life: Prompt identification and action to improve the lives of those who are impacted and those who are caring for them.
- Enabling personalized medicine by using predictive models to generate treatment strategies that are specifically customized to meet the needs of each patient.
- Innovation Promotion: Using machine learning to anticipate Alzheimer's illness and produce new insights and remedies.
- beneficial Impact: Improving the early diagnosis, prediction, and treatment of Alzheimer's disease is the goal, with the intention of having a beneficial impact on people, families, and society.

Problem:

Problem Analysis:

Millions of people worldwide suffer from the complicated and debilitating disorder known as Alzheimer's disease. It accounts for 60–80% of dementia cases, making it the most frequent cause. The disease worsens over time, making it harder for the patient to do daily tasks and ultimately leading to total dependence on caregivers. The biggest risk factor for Alzheimer's disease is getting older; most instances affect those over 65. According to estimates from the World Health Organization (WHO), 50 million individuals worldwide suffer from dementia, with Alzheimer's disease accounting for many of these instances.

Objective:

Develop a machine learning model to assist in the early detection and progression monitoring of Alzheimer's disease, with the aim of improving patient outcomes, enhancing caregiving strategies, and advancing our understanding of the disease.

Review and analysis of current existing techniques for Alzheimer's disease.

Implementation of novel machine learning algorithm for Alzheimer's disease detection.

Comparative analysis of proposed machine learning model with existing techniques of Alzheimer's disease detection.

Conclusion:

Conclusively, Alzheimer's disease is a significant global concern that impacts individuals, families, and healthcare systems in equal measure. To facilitate timely interventions, improving patient outcomes, reducing caregiver loads, and lowering healthcare costs, early diagnosis and accurate disease prediction are essential. Strong tools for building prediction models that identify people at risk of Alzheimer's disease onset even before symptoms appear are provided by machine learning algorithms. Predictive modeling is used not only in clinical settings but also in public health campaigns, policy development, and community outreach. To progress the prediction of Alzheimer's disease and turn research findings into useful applications, cooperation between researchers, physicians, legislators, and industry stakeholders is essential.

Chapter 2: Review of Literature

A Comparative Analysis of Machine Learning Algorithms to Predict Alzheimer's Disease, a journal paper released in 2021 [1], discussed the accuracy of the algorithms based on several algorithms that the authors worked on. Their employed algorithms, with corresponding accuracies of 92%, 74.7%, 80%, and 81.3%, are Support Vector Machine, Logistic Regression, Decision Tree, and Random Forest. The highest accuracy is on the SVM algorithm, which is 92%, shows the best accuracy in this journal paper. They took the datasets from the Kaggle and OASIS platform for their project. Their research paper says that in future, models should be improved by using large and multiple datasets and implementing other algorithms such as AdaBoost, KNN, Bagging and Majority Voting. Increasing performance of the system will increase the reliability of providing the best results. These Machine learning systems give an idea about the patients that they are having the disease just by putting the MRI data in the implemented algorithm. These algorithms may help the public to identify their disease at initial stages only.

The journal paper 'Primate brain pattern-based automated Alzheimer's disease detection model using EEG signals' was published in the year of 2022 [2]. Here the EEG means Electroencephalography which may detect early changes in Alzheimer's disease. They have collected the dataset from one of the platforms like Datashare. The KNN algorithm, which was applied as a loss vector calculator in an iterative feature selection and classifier process, served as the foundation for the model classification. They have divided the ratio into Tenfold CV and LOSO CV using the Primate brain pattern as their method. In Tenfold CV the accuracy is 100% and in LOSO CV they have an accuracy of 92.01%. For improved outcomes, their model incorporates multi-level feature extraction, iterative feature selection, classification, and IMV. The model they have adopted is simple to use and works well with common gear. The graph-based outcomes of this model greatly encourage other researchers to test out novel strategies or alternative algorithms using other datasets.

The journal paper ‘Machine Learning and Novel Biomarkers for the Diagnosis of Alzheimer’s Disease’ was published under the international journal of Molecular Science in the year 2021 [3]. In this research paper, the researchers have used many algorithms to find the best accuracies. The algorithms they have used are Ensemble-learning-73%, Regression Tree-92%, CNN-90.8%, SVM-68%, Logistic Regression, RF-85%, Naïve Bayes, Deep Learning-85%, and Extreme Gradient Boosting-88%. The highest accuracy they have got is Deep Learning CNN which is 90.8%. They have collected the data in the forms of images from different sources like ADNI, AIBL, OASIS and MIRIAD. They took around 8800+ images to work on the model and provide the highest result. Moreover, no one can promise such results using machine learning, but increasing the training data and variety of data with a new approach is always there. Most of the time increasing studies and working on the new technology with biomarkers as promising approaches to predict the disease.

The journal paper ‘Review and Comparison for Alzheimer’s Disease Detection with Machine Learning Techniques’ was published under International Neurology Journal in the year of 2023 [4]. In this research paper, they have worked on multiple algorithms to find the highest accuracy. The classification and techniques they have used are SVM, Decision Tree, NB, and RF. They have collected the data from the Global Dementia Observatory which is created by the World Health Organization (WHO). They took some previous works and made some comparisons and provided the results. The highest accuracy they have got till the date is 95% using RNN algorithm. In the field of neurology, the use of machine learning techniques is crucial for developing strategic models and for early detection of Alzheimer's disease. The objective is to acquire knowledge and the capacity to treat fatal illnesses, and the future breadth of the ML-based detection model is being enhanced.

The paper ‘A Comprehensive Machine-Learning Model Applied to Magnetic Resonance Imaging (MRI) to Predict Alzheimer's Disease in Older Subjects’ was published under the Journal of Clinical Medicine in

the year of 2020 [5]. They have used 4 different classifiers to predict the disease. The models they have used with predictions are Naïve Bayes-88.76%, ANN-83.56%, 1NN-91.32%, SVM-89.67% as Autonomous Feature Selection. The model performance increased significantly after the feature selection process, they increased by 2-3% on average. Accuracies after feature selection are Naïve Bayes-93.44%, ANN-83.56%, 1NN-95.92% and SVM-96.12%. In the future scope and limitations, they have mentioned disease detection in which they find it very difficult to predict the disease at a very early stage and there are no such tools or models for its simple detection. In future research they are considering both genetic and nongenetic features for the hybrid detection machine learning model.

Using T1-weighted MRI scans, Jong Bin Bae et al. [2020] looked at the application of a convolutional neural network (CNN) for Alzheimer's disease (AD) classification [6]. For the ADNI dataset, the model's accuracy was 88%, while for the SNUBH dataset, it was 89%. Two separate populations with different ethnic backgrounds and educational attainment were engaged in the study. The CNN model used 2D MRI slices focusing on the medial temporal lobe, a region commonly affected in AD. This approach achieved fast processing times and maintained high accuracy regardless of the population. The study highlights the potential for generalizability of CNN models across diverse populations for AD detection.

Article titled "Comparing different algorithms for the course of Alzheimer's disease using machine learning," Tang et al. (2021) investigated the effectiveness of various machine learning algorithms for classifying AD progression [7]. They utilized data from the ADNI database, focusing on 560 subjects categorized into four groups: cognitively normal, early mild cognitive impairment, late MCI (LMCI), and AD. The study employed three machine learning techniques: RF, SVM, and DT. MRI features were fed into these algorithms to predict the disease progression (CN-EMCI-LMCI-AD). The results revealed that the RF classifier achieved the highest accuracy (73.8%) compared

to SVM (60.7%) and DT (59.5%). Additionally, the RF model demonstrated a superior Area Under the Curve of 0.92 for CN-AD prediction. The study suggests that the RF classifier, along with a selection of five optimal MRI features, holds promise for early-stage AD classification, potentially aiding in diagnosis.

Rashmi Kumari (2020) published work named “Machine learning technique for early detection of Alzheimer’s disease”, a CNN achieved an accuracy of 90.25% for early detection of AD [8]. The model utilized data from the ADNI and OASIS datasets. This approach outperformed a KNN classifier, demonstrating the effectiveness of CNNs for AD detection. The study highlights the potential of machine learning for early and accurate AD diagnosis.

Using multiclass classification to diagnose Alzheimer's disease in its early stages seems to be a difficult undertaking because the findings are often average when it comes to AD stage detection. The real-time deep and transfer learning features and classification techniques that we have shown in this work effectively identify the multiclass categorization of Alzheimer's disease. We employed a pre-learned AlexNet network for transfer learning assisted deep feature detection. We adjusted and refined these models to satisfy the demands of our issue. Textural and statistical features make up handcrafted features. SVM, KNN, and RF are employed as classifiers for the assessment of the handmade feature extraction model and the deep features model. With 92.85% accuracy for deep learning CNN and 99.21% accuracy for a deep feature, we reached the best results. The findings suggested that, when compared to alternative approaches, research using transfer learning models appeared to perform well. Due to their extensive dataset pre-training, these models have the best accuracy, which is reflected in their network performance. The models that we suggested produced encouraging outcomes. Because of their excellent multiclass classification accuracy for the early identification of Alzheimer's disease [9].

Koga et al. (2020) presented a deep learning model for differentiating tauopathies [10]. Their approach combined object detection and a RF

classifier achieving an accuracy of 97% on a dataset of tau-immunostained digital slides from Mayo Clinic. The model distinguished AD, Progressive Supranuclear Palsy, Corticobasal Degeneration, and Pick's disease based on lesion types in CP13-stained slides. Validation using AT8-stained slides confirmed model efficacy, suggesting potential generalizability to other phospho-tau antibodies. This work shows how deep learning can be used to analyse tau pathology objectively and quantitatively, which could help pathologists and enhance clinicopathological correlations. Nevertheless, single-centre training and the requirement to include more tauopathy kinds and staining techniques for wider applicability are drawbacks.

The application of an ensemble learning approach for Alzheimer's disease classification is highlighted in the journal publication "Deep ensemble learning for Alzheimer's diseases classification" by Ning An, which was published on March 23, 2020 [11]. This paper demonstrates the application of deep learning algorithms for the classification of Alzheimer diseases. Deep ensemble learning is a technique that integrates multisource data and leverages its advantages to leverage the strengths of deep learning. Essentially, ensemble learning uses multiple algorithms to obtain better predictive analysis than the use of single algorithms. In this research the data for classification is obtained from national Alzheimer's coordinating centres and its application shows an improved accuracy of 4% that is shown to be better from other 6 well known ensemble learning frameworks. The paper proposes an advance approach to the classification and detection Alzheimer diseases. This research uses multiple algorithms, and every algorithm is divided into feature spaces of A, B and C and shows different accuracies with least ranging to 63.1% using stacking classifier and highest ranging to 80.1% using RF classifiers.

The journal paper "Convolutional neural network for Alzheimer's diseases detection on MRI images," published on 29th April 2021 by Amir Ebrahimi talks about the use of CNN on MRI to detect AD [12], the use of CNN is shown to be useful for detecting AD in its preliminary

states. Their approach to this study includes implementation and comparisons of several deep model and configurations using two dimensional and three dimensional CNNs and have also used the method of recurrent neural network , the approach consists of splitting MRI scans into two 2D slices , in this approach the RNN can understand the connection between the two splits of the 2D slices of the MRI, the main contribution of this study is to introduce transferring learning from a dataset of 2D images to 3D CNNs. This method has shown improved accuracy of sliced method by 2% and an overall accuracy of 96.88% and 100% of sensitivity. This research has shown that transfer learning from ImageNet to MRI datasets shows better results compared to other methods.

Research article on “The Machine Learning and Image Processing Enabled Evolutionary Framework for Brain MRI Analysis for Alzheimer’s Disease Detection” published on 28th March 2022 by Mustafa Kamal aims to identify the protein differences and understand the formation of brain nerves, this research uses machine learning algorithms to detect the diseases. Researchers have used the techniques like image denoising and histogram Equalising, the classification algorithms used in this research are LS-SVM, SVM, KNN, and RF Classifier, the accuracy in this research paper after using SVM was 86.2%, and using KNN was 95.833%. A graphical representation showed that LS-SVM showed the highest accuracy of 98% while the KNN method showed the highest sensitivity overall the LS-SVM method showed better accuracy and better precision over other methods.

Research paper on “Alzheimer’s Disease Diagnosis using Deep Learning Techniques” published on 3rdFebruary by Ahmad Waleed Salehi shows the use of deep learning in the identification of Alzheimer’s diseases [13], in this research the authors have conducted a thorough research on different deep learning algorithms used worldwide on Alzheimer’s diseases detection. The study combines data from various sources such as ADNI and OASIS to improve their predictive analysis, the researchers proposes that use of advance deep learning

algorithms in combination of diverse datasets from various sources could increase the accuracy of the classification. The researchers have used the CNN classifiers in diagnosis of Alzheimer's diseases using MRI Images. The research shows an accuracy of 98.94% in the identification of the Alzheimer's diseases.

Research paper on "A Machine Learning Approach for Differential Diagnosis and Prognostic Prediction in Alzheimer's Disease" published in year 2023 by Balram Yadav Kasula presents a machine learning approach to tackle complex challenges of diagnosis and prognosis of Alzheimer's Diseases [14]. To provide a comprehensive understanding of diseases through various imaging analyses, this paper also used a variety of datasets, including neuroimaging scans, clinical evaluations, and demographic information from both AD patients and healthy controls. The research demonstrates accuracy and specificity from diagnosis performance and prognostic views, with accuracy being shown to be 90% in model 1, 88% in model 2, and 91% in model 3.

The Research journal paper 'An Improved Multi-Modal based Machine Learning Approach for the Prognosis of Alzheimer's disease' which was published in 2022 [15]. They used Random Forest algorithms, which have accuracy rates as high as 87%. For their research, they used datasets from the OASIS platform. According to their research report, machine learning models may prove to be extremely important in the future for assessing how well AD treatment approaches work. and using ML approaches to the diagnosis of AD is a promising way to further our knowledge of the illness and enhance patient care. Though encouraging outcomes have been obtained so far, AD diagnosis models still require development and enhancement.

The Research journal paper 'Two-stage deep learning model for Alzheimer's disease detection and prediction of mild cognitive impairment time' which was published in the year of 2022 by Shaker El-Sappagh have mentioned their accuracies based on different algorithms which they have worked on [16]. The algorithms they have used are DT, RF, SVM, LR, and KNN which have accuracies like

82.82%, 92.01%. They took the datasets from the ADNI platform for their project. Their research paper says that in future, Exploring the integration of additional modalities, such as genetic markers, lifestyle factors, and environmental exposures, can enhance the predictive accuracy and comprehensiveness of the framework. Despite its limitations, the study sets a foundation for future research endeavours aimed at enhancing model interpretability, incorporating additional modalities, and extending the applicability of the proposed framework to diverse domains.

The research journal article "A Machine Learning Model to Predict the Onset of Alzheimer Disease using Potential Cerebrospinal Fluid Biomarkers," written by Syed Asif Hassan and published in 2017, discussed the accuracy of the model based on various algorithms that the authors had developed [17]. They have generated J48, NB, and SMO algorithms. Which exhibit accuracy levels of 98.82% For their project, they used datasets from the Kaggle platform. Considering the early-stage biomarker qualities of the clinical Alzheimer data, their research paper recommends the implementation of an online prediction tool in the future to screen people for early-stage cognitive impairment. The results highlight the ability of ML algorithms to improve diagnostic precision and identify people who are at risk of AD progression. The J48-based model's excellent sensitivity, specificity, and accuracy indicate its usefulness in clinical settings for detecting patients who are at risk of AD progression.

The Research journal paper 'Machine Learning and DWI Brain Communicability Networks for Alzheimer's Disease Detection' which was published in the year of 2020 by Eufemia Lella have mentioned their accuracies based on different algorithms which they have worked on [18]. The algorithms they have used are SVM, RF, and ANN. have accuracies like ANN 0.75 ± 0.01 . this study addresses two critical issues in Alzheimer's disease diagnosis and brain network analysis They took the datasets from the ADNI platform for their project. Their research paper says that in the future Investigating the progression from MCI to

AD offers valuable insights into disease dynamics and treatment efficacy. the groundwork for future research aimed at improving diagnostic accuracy and clinical decision-making in neurodegenerative diseases.

The Research journal paper ‘Multimodal deep learning models for early detection of Alzheimer’s disease stage’ which was published in the year of 2021 by Janani Venugopalan have mentioned their accuracies based on different algorithms which they have worked on [19]. The algorithms they have used are DT, RF, SVM, LR, and KNN which have accuracies like 79%, 83%,81%,71%. They took the datasets from the ADNI platform for their project. Their research paper says that the future use of the developed DL model holds significant potential to transform AD diagnosis, prognosis, and treatment strategies.

The journal paper ‘Prediction and Classification of Alzheimer’s Disease using Machine Learning Techniques in 3D MR Images’ was published by Kongala Nageswara Rao in the year 2023 [20]. In this research paper, the researchers have used many algorithms to find the best accuracies. The algorithms they have used are CNN:94.23%, SVM & MLP:97.47%. The highest accuracy they have got is Deep Learning SVM & MLP which is 97.47%. They have collected the data in the forms of images from 3D Magnetic Resonance Imaging (MRI) scans. Limitations of this paper include limited dataset size and challenges with interpretability. Based on MRI scans, the machine learning models are useful in properly diagnosing and classifying Alzheimer's disease based on a high accuracy and balanced Precision, Recall, and F1-Score.

The journal paper ‘Brain MRI Image Analysis for Alzheimer’s Disease Diagnosis Using Mask R-CNN’ was published by Madhuri Under in the year 2024 [21]. In this research paper, the researchers have used many algorithms to find the best accuracies. The algorithms they have used are Mask R-CNN: 97.46%. They have been using the ADNI dataset. It has limitations in that the generalizability and robustness of the model would be improved with a larger and more varied dataset. Furthermore,

computerized diagnosis and therapy monitoring have promising future paths indicated by Alzheimer's disease.

This study utilized deep learning and transfer learning techniques on a large and diverse MRI dataset to develop a highly accurate Alzheimer's disease (AD) diagnosis classifier. By training on 85,721 brain scans from 50,876 subjects across 217 sites/scanners, the model achieved a 90.9% accuracy rate in cross-validation and 94.5%/93.6%/91.1% accuracy on three unseen datasets. Notably, it predicted AD status in individuals with mild cognitive impairment (MCI) more frequently than those who did not convert, suggesting potential for early detection. The model's predicted scores correlated significantly with disease severity, indicating its potential integration into clinical practice as a valuable diagnostic tool. Bin Lu et al (2022) [22].

This study explored the use of neuroimaging biomarkers, specifically structural MRI images, for diagnosing dementia and Alzheimer's disease. By employing CNN models trained on pre-processed brain images, the study achieved an accuracy of approximately 80% in diagnosing Alzheimer's disease and moderate cognitive impairment. However, diagnosing mild cognitive impairment proved more challenging. The findings suggest that deep learning-based models, when integrated with other clinical tests, could serve as valuable diagnostic tools for Alzheimer's disease and related conditions. A. Yigit et al (2020) [23].

This paper introduces a new method for diagnosing Alzheimer's disease (AD) and detecting its progression, addressing common research challenges. By combining 11 modalities from the ADNI dataset, the model achieves high accuracy and interpretability. It is composed of a two-layer random forest classifier, where the first layer detects the transition of MCI to AD within three years, while the second layer diagnoses early AD. Selected markers are used to maximize the model's performance. Emphasis is placed on explainability; 22 explainers based on fuzzy rule-based systems and decision trees, as well as global and instance-based explanations supplied using the SHAP framework, are

included. These explanations, which are given in plain English, help doctors understand. In the first layer, the model achieves 93.95% accuracy and an F1-score of 93.94%; in the second layer, it achieves 87.08% accuracy and an F1-score of 87.09%. It improves clinical comprehension and offers insights that are in line with AD literature. It is reliable and accurate. Shaker and associates (2021) [24].

Alzheimer's disease is a chronic illness that affects many older people and poses serious health risks. Deep learning has become the go-to method for medical image analysis, particularly AD detection, in recent years due to its impressive success and rise in popularity. This work investigates biomarkers associated with AD, investigates feature extraction approaches, examines the application of deep learning methods in AD detection, and synthesizes and analyses several AD detection models and methods. The results highlight the superior accuracy and efficiency of deep learning technology over traditional machine learning techniques in AD detection. Gao and associates (2022) [25].

Alzheimer's Disease (AD) is an incurable neurodegenerative condition, emphasising the importance of early detection for effective management. Neuroimaging, particularly Positron Emission Tomography (PET), aids in diagnosis by providing detailed brain images. To enhance diagnostic accuracy, a new Computer-Aided Diagnosis system based on CNN was developed. Tested on 855 patients' 18FDG-PET images from the ADNI database, the CAD system achieved impressive results: 96% accuracy, 96% sensitivity, and 94% specificity. This outperforms previous methods, highlighting its potential for improving AD diagnosis and patient care. Ahila . A et al (2022) [26].

TABLE I. LITERATURE REVIEW SUMMARY

REFERENCE	ACCURACY	DATASET	LIMITATIONS
[1]	SVM-92%, LR-74.7%, DT-80%, RF-81.3%	KAGGLE & OASIS	OVERFITTING
[2]	DATA-SHARE	KNN-92%	DIVERSE POPULATIONS AND POTENTIAL CHALLENGES IN INTERPRETING FEATURES OF DATASET
[3]	EL-73%, RT-92%, CNN-84.2%, DL_CNN-90.8%, SVM-68%, DL-85%, XGBOOST-88%, RF-85%	ADNI, AIBL, NACC	MULTIDOMAIN INTERVENTIONS MAY BE BURDENSOME AND NOT UNIVERSALLY ACCEPTABLE
[4]	SVM, NN, DT-79%, RNN-95.6%, SVM-86%, SVM, RF-87-94%, NB-68%	GLOBAL DEMENTIA OBSERVATORY	ENABLING PROMPT INTERVENTION AND BETTER PATIENT OUTCOMES
[5]	NB-93.44%, ANN-83.56%, KNN-95.92%, SVM-96.12%	NA	TO IDENTIFY AD AT VERY EARLY STAGES IS DIFFICULT
[6]	CNN_ADNI-88%,	ADNI & SNUBH	MAY NOT CAPTURE NUANCES OF ATROPHY IN OTHER

	ANN_SNUBH-89%		BRAIN REGIONS, POTENTIALLY LEADING TO MISCLASSIFICATION
[7]	RF-73.8%, SVM-60.7%, DT-59.5%	ADNI	SAMPLE SIZE AND PREDICTION ACCURACY IMPROVEMENT
[8]	CNN-90.25%	ADNI & OASIS	COMPARISON WITH PREVIOUS STUDIES IS LIMITED
[9]	SVM-99.21%, KNN-57.32%, RF-93.97%	OASIS	MULTI-CLASSIFICATION MODEL ACHIEVES ORDINARY RESULTS
[10]	RF-97%	DAY CARE (MAYO)	RF COULDN'T RECOGNIZE THIS CASE PSP
[11]	RF-80.1%, NB-75.8%	NACC	INCONSISTENCIES IN THE DATA
[12]	2D-3D CNN's-96.80%	MNIST OR IMAGENET	RESTRICTED KNOWLEDGE IN IDENTIFYING THE PARTS OF THE BRAIN AFFECTED BY AD
[13]	SVM_ADNI-100%, SVM_OASIS-97%	ADNI & OASIS	POTENTIAL FOR IMPROVED EARLY-STAGE AD PREDICTION WITH ADVANCED DEEP LEARNING ON

			COMBINED DATASETS
[14]	SVM-90%, CNN-88%, LSTM-91%	ADNI	CHALLENGES IN DATA HETEROGENEITY AND MODEL INTERPRETABILITY NECESSITATE FURTHER EXPLORATION
[15]	RF-87%	OASIS	ADVANCEMENT AND BROAD IMPROVEMENTS REQUIRED
[16]	DT-82.82% RF-92.01%	ADNI	MEDICAL EXPERTS DON'T TRUST ML DECISION WITHOUT AN ACCURATE EXPLANATION
[17]	J48-96.92%, NB-76.92, SMO-87.70%	KAGGLE	SOLE RELIANCE ON CSF BIOMARKERS IGNORES OTHER FACTORS INFLUENCING COGNITIVE IMPAIRMENT
[18]	SVM-72.5%, RF-74%, ANN-77%	ADNI	NEEDS MORE DATA AND EXPLORING OTHER WAYS TO IMPROVE ACCURACY

[19]	SVM-79%, RF-83%, DT-81%, KNN-71%	ADNI	LARGE DATA IS NEEDED TO VALIDATE THE DL MODELS
[20]	CNN:94.23%, SVM & MLP:97.47%	(BRFSS)	INTERPRETABILITY CHALLENGES & LIMITED DATASET SIZE
[21]	MASK R-CNN: 97.46%	ADNI	SMALL DATASET, LIMITED DISEASE TYPE, SINGLE EVALUATION METRIC
[22]	INCEPTION-RESNET-V2-94.9%	ADNI, AIBL, MIRIAD, OASIS	SOLE USE OF STRUCTURAL MRI DATA, RELIANCE ON ADNI DATABASE FOR LABELS
[23]	CNN-80%	STATISTA & OASIS	DIAGNOSIS OF MILD COGNITIVE IMPAIRMENT IS DIFFICULT THAN DIAGNOSING AD
[24]	RF-93.95%, SHAP-93.94%	ADNI, AIBL, MIRIAD, OASIS	VALIDATION NEEDED ON DIVERSE DATASETS
[25]	MiFi-90%	ADNI, OASIS, AIBL	OVERFITTING AND, LIMITED MEDICAL DATA
[26]	CNN-96.8%	ADNI	DEPENDENCY ON 18FDG-PET IMAGES RESTRICT APPLICABILITY

Chapter 3: Implementation of the Project

In the Alzheimer's disease (AD) detection study, we applied a research design and methodology focusing on using convolutional neural networks (CNN) and recurrent neural networks (RNN) for the model of us. Alzheimer's disease (AD) is a neurodegenerative disorder characterized by progressive cognitive decline, memory loss, and behavioural changes. It is known to be the most common cause of dementia in older adults.

Convolutional neural networks (CNN) and recurrent neural networks (RNN) are two types of artificial neural networks used in machine learning and deep learning implementations. Convolutional Neural Network (CNN):

- **Convolutional Neural Networks (CNNs):**
 - CNNs are specifically used for effective image recognition of images and tasks task classification, making them useful in the process of medical imaging image analysis in the field of medical innovation driven by artificial intelligence driven medical innovation, intelligence, including the detection and diagnosis of AD. CNNs are seen shown to be outperforming at perform superiorly in learning hierarchical representations of images by applying convolutional filters, which filters that can capture spatial dependencies and patterns in the data.
- **Recurrent Neural Networks (RNNs):**
 - RNNs are designed to process sequences of data, best use for time-series analysis, and NLP (natural language processing), and sequential data tasks. Talking in the context of AD detection, RNNs can be applied to the data to analyse temporal patterns or sequential data related to patients' cognitive assessments and reports, such as changes in memory pattern of the patients and their detreating language abilities over the time. RNNs have been widely used in AD research to model progression of the disease based on patient's data, their cognitive test results, and other relative data of the patient.
 - CNNs and RNNs offer supplementary strengths for analysing different types of data relevant to AD detection. While CNNs outperforms at

extracting features from imaging data, RNNs are well used for modelling temporal patterns in cognitive reports. Combining these neural network architectures into a single model can enhance the accuracy and effectiveness of AD detection systems, ultimately contributing to early diagnosis and improved patient outcomes.

- **Software Framework**

- The implementation is done out with TensorFlow, an open-source machine learning framework. TensorFlow provides broad support for developing and training deep learning models, allowing for fast execution on both CPUs and GPUs.

Problem objective:

- Our research aimed at developing a machine learning model that uses brain MRI images and medical imaging data to identify the Alzheimer's disease (AD).
- Our main aim of research was using CNN and RNN to determine its efficiency and accuracy in detecting Alzheimer's disease.

Data collection:

- Data for training our model was collected from multiple sources. We collected brain MRI images and medical imaging data from multiple sources and used some publicly available dataset from OASIS and Kaggle.
- The dataset contained MRI images of brain of different people.

Data Pre-Processing:

- Before the model development stage an extensive data pre-processing was performed to avoid any error during the development stage.
- This Pre-Processing of data included handling any missing values in the dataset and normalizing the MRI images to enhance its quality.
- Data Augmentation technique was also implemented during this stage.

Model Building/Development:

- We constructed two different neural network techniques CNN and RNN.
- Model architecture was designed using these two neural network techniques and the process of compilation and training was implemented.

Training and validation:

- After the development stage, we started the model training and validation of our model.
- This stage considered of splitting our datasets into training split and testing split to evaluate model's accuracy and prediction comprehensively.

Model evaluation metrics:

- Performance evaluation of model was done using standard metrics such as accuracy and precision.
- This analysis and performance metric allowed us to draw conclusions about the efficiency of our model.
- Our model showed accuracy of 100% using CNN architecture model and 97.33% using the RNN architecture model.

CNN – Layers Breakdown

```
def conv_block(filters):  
    block = tf.keras.Sequential([  
        tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),  
        tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),  
        #tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', strides=2, padding='same'),  
        tf.keras.layers.BatchNormalization(),  
        tf.keras.layers.MaxPool2D()  
    ])   
  
    return block
```

Figure 1 -Code snippet for Separable Convolutional Block with Batch Normalization & MaxPool2D

The function conv_block shown in fig() describes a convolutional block that is commonly used in Convolutional Neural Networks (CNNs) to perform tasks such as image categorization and object detection. Let Us Explore it further:

- **`tf.keras.layers.SeparableConv2D`**: This layer separately applies spatial convolutions to every channel. The ReLU activation function and padding are used after it to preserve the spatial dimensions.
- **`tf.keras.layers.BatchNormalization`**: By normalizing the activations of the preceding layer, this layer facilitates more effective and efficient training of deep neural networks.
- **`tf.keras.layers.MaxPool2D`**: By executing the max pooling procedure, this layer lowers the input volume's spatial dimensions.

Thus, the layers employed in this case are Batch Normalization, Max Pooling, and Separable Convolution. In CNN designs, these layers are frequently utilized for feature extraction and spatial dimension reduction.

```
def dense_block(units, dropout_rate):  
    block = tf.keras.Sequential([  
        tf.keras.layers.Dense(units, activation='relu'),  
        tf.keras.layers.BatchNormalization(),  
        tf.keras.layers.Dropout(dropout_rate)  
    ])   
    return block
```

Figure 2 – Dense Block with Dropout and Batch Normalization

The dense block function creates a block comprising a dense layer, followed by batch normalization and dropout regularization. Let us breakdown its layers:

Dense Layer: `tf.keras.layers` is the dense layer. Dense: The fundamental component of a fully linked neural network is this layer. It is made up of an equal number of neurons, each of which is coupled to every other neuron in the layer above. ReLU (`activation='relu'`) is the activation function that is employed in this instance. It adds non-linearity to the network, enabling it to recognize intricate patterns in the data.

Dropout Regularization: `'tf.keras.layers.Dropout'` A regularization approach that reduces the model's dependence on neurons during training.

RNN- Layer Breakdown

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import SimpleRNN, Dense  
  
def build_rnn_model(input_shape):  
    model = Sequential([  
        SimpleRNN(50, activation='relu', input_shape=input_shape, return_sequences=True),  
        SimpleRNN(20, activation='relu'),  
        Dense(64, activation='relu'),  
        Dense(4, activation='softmax')  
    ])   
    return model  
  
# Assuming train_data_seq is reshaped to (-1, img_height, img_width * channels)  
input_shape = (train_data_seq.shape[1], train_data_seq.shape[2]) # (img_height, img_width * channels)  
  
model = build_rnn_model(input_shape)  
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
model.summary()
```

Figure 3 – Sequential RNN and Dense Layers Setup

SimpleRNN Layer (50 units): This layer contains 50 units With ReLU activation function, that takes accepts inputs as defined input shapes. ``return_sequences`=True` is set to return sequences instead of a single output at each time step.

SimpleRNN Layer (20 units):

- This layer contains 20 recurrent units with ReLU activation function.
- operates on the output sequences of the previous SimpleRNN layer.

Dense Layer (64 units):

- A fully connected layer with 64 units and ReLU activation function.
- operates on the output of the second SimpleRNN layer.

Dense Layer (4 units):

- The final output layer with 4 units (assuming a multi-class classification task) and SoftMax activation function.
- It generates the probability distribution across the various classes.

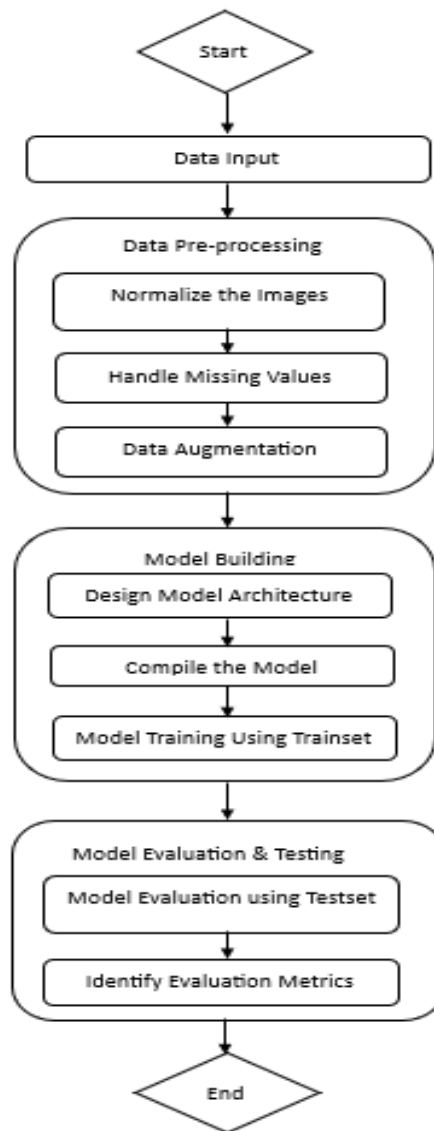


Figure 4 – Model Architecture of our Project.

Chapter 4: Result and Discussions

We've conducted our project using two distinct models and datasets sourced from platforms such as Kaggle and OASIS.

The Kaggle dataset offers pre-organized data sets, conveniently divided into trainsets and test-sets folders. Within these sets, the data is segmented into four distinct categories: pituitary, no-tumor, meningioma, and glioma. Each category represents a different pathological condition affecting the brain, each with its own set of implications for cognitive health and potential connections to Alzheimer's disease. Pituitary tumors, for instance, are linked to hormone regulation, as the pituitary gland performs a vital position on this process. Dysfunction in this gland can disrupt numerous physical functions, such as pressure response, energy regulation, and reproductive health, all of which have indirect connections to Alzheimer's disease progression. For instance, alterations in pituitary function may exacerbate symptoms related to Alzheimer's, such as changes in appetite, sleep patterns, and mood. Conversely, the absence of a tumor (no-tumor category) simplifies the diagnostic process for Alzheimer's disease. Without the additional complexity of managing a tumor, healthcare providers can focus more directly on assessing and treating Alzheimer's-related symptoms, leading to a clearer diagnosis and more targeted interventions. Meningiomas and gliomas, although differing in their cellular origins and aggressiveness, can both complicate the analysis and control of Alzheimer's ailment due to their effects on cognitive function. Meningiomas, arising from the meninges, can exert stress on surrounding brain tissue, main to signs and symptoms together with headaches, seizures, and cognitive impairment. Similarly, gliomas, originating from glial cells, can disrupt neural pathways and cognitive functions, mimicking or exacerbating Alzheimer's symptoms. The presence of these tumors can make it challenging to distinguish between Alzheimer's-related cognitive decline and tumor-induced symptoms. Additionally, the treatment approaches for both conditions may interact and require adjustment to address the unique challenges

posed by each. Therefore, a coordinated and comprehensive care approach is essential for effectively managing both the cognitive symptoms of Alzheimer's ailment and the tumor-related complications, ensuring the best possible outcomes for patients.

We leveraged a variety of libraries including TensorFlow, NumPy, Matplotlib, Scikit-learn, Seaborn, and Keras to handle image reading and perform analyses on diverse datasets. Through the utilization of these resources, we were able to efficiently process and interpret the data, enabling us to extract valuable insights.

```
In [1]: 1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.metrics import classification_report, confusion_matrix
5 import seaborn as sn
6 from tensorflow.keras import backend as K
7 from tensorflow.keras.models import Sequential, Model, load_model
8 from tensorflow.keras.layers import Dense, Flatten, Dropout, Activation, Conv2D, MaxPooling2D, LeakyReLU
9 from tensorflow.keras.callbacks import TensorBoard
10 import pickle
11 from PIL import Image
12 import time
13 import os
14 from PIL import Image
15 from tensorflow.keras.regularizers import l2
16 from sklearn.model_selection import train_test_split
17
18 try:
19     tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
20     print('Device:', tpu.master())
21     tf.config.experimental_connect_to_cluster(tpu)
22     tf.tpu.experimental.initialize_tpu_system(tpu)
23     strategy = tf.distribute.experimental.TPUStrategy(tpu)
24 except:
25     strategy = tf.distribute.get_strategy()
26
WARNING:tensorflow:From C:\Users\RITU PRIYA SINGH\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Figure 5 – Importing necessary libraries and TPU Configuration for our model implementation.

In the Kaggle dataset, we defined precise parameters for data analysis. These included setting the image size to [176, 208], a batch size of 5000, with a 10% test split and a 20% validation split. We applied a zoom range of [0.99, 1.01] and a brightness range of [0.8, 1.2]. Furthermore, we unfroze layers, configured a learning rate to 0.01, batch settings of 20, and conducted training for 20 epochs.

```
In [2]: 1 IMAGE_SIZE = [176,208]
2 BATCH_SIZE = 5000
3 test_split_percent = .1
4 validation_split_percent = .2
5 zoom = [.99,1.01]
6 bright_range = [.8,1.2]
7 layers_unlocked = True
8 lr = 0.01
9 batch = 20
10 EPOCHS = 20
```

Figure 6 - Deep Learning Hyperparameters

We applied image augmentation techniques, specifically zoom and brightness adjustments, to both the training and testing datasets for image preprocessing. Using TensorFlow's Keras library, we employed an ImageDataGenerator with defined parameters for rescaling, constant fill mode, brightness range, and zoom range to process the training data. The zoom range accommodates variations in image size and orientation, while adjusting brightness aids in handling diverse lighting conditions. These transformations aim to enhance the model's robustness and generalization capabilities by introducing variability. During training, data is loaded from the specified directory, resized to a target image size, and organized into batches. For testing, a separate ImageDataGenerator is created without zoom adjustments to maintain the original size and proportions of the test data. This ensures a realistic evaluation of the model's overall performance on unaltered data. The testing data generator loads data from the specified directory, maintains the target image size and batch size, and preserves the original data order by not shuffling it. These techniques optimize the model's performance while upholding the integrity and consistency of the testing data, facilitating a thorough assessment of the model's accuracy and effectiveness.

```
In [3]: 1 train_dr = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255, fill_mode='constant', cval=0,
2                                     brightness_range=bright_range, zoom_range=zoom,
3                                     data_format='channels_last', zca_whitening=False)
4
5 train_data_gen = train_dr.flow_from_directory(directory="./archive/Training/", target_size=IMAGE_SIZE,
6                                             batch_size=BATCH_SIZE)
7
8 test_dr = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255, fill_mode='constant', cval=0, zoom_range=[1,1],
9                                     data_format='channels_last')
10 test_data_gen = test_dr.flow_from_directory(directory="./archive/Testing", target_size=IMAGE_SIZE, batch_size=BATCH_SIZE,
11                                           shuffle = False)

Found 5712 images belonging to 4 classes.
Found 1311 images belonging to 4 classes.
```

Figure 7- Image Data Generators Setup

Following preprocessing, the next method is invoked to obtain batches of training and testing data, along with their respective labels. The training data comprises images pre-processed based on specified parameters like brightness and zoom range, accompanied by train labels representing expected outputs for model learning. Conversely, the testing data remains unaltered to preserve its original form, while test labels provide known outputs for model evaluation. This approach

guarantees effective learning from augmented training data and accurate assessment using realistic testing data.

```
In [4]: 1 train_data,train_labels = train_data_gen.next()
        2 test_data,test_labels = test_data_gen.next()
```

Figure 8 - Fetching Batch of Images and Labels

The concatenate method is utilized to merge both the training and testing datasets along with their respective labels. This consolidation facilitates the aggregation of all data into single arrays, streamlining subsequent processing and analysis steps. Specifically, the function combines the training and testing data into a unified array named `total_data`, encompassing all images from both sets. Likewise, it merges the training and testing labels into another array named `total_labels`, incorporating all expected outputs corresponding to the images in `total_data`.

```
In [5]: 1 total_data = np.concatenate((train_data,test_data))
        2 total_labels = np.concatenate((train_labels,test_labels))
        3 print(total_data.shape)
        4 print(total_labels.shape)

(6311, 176, 208, 3)
(6311, 4)
```

Figure 9- Concatenate Train and Test Data

The code snippet performs a two-step data splitting process to perform the division of the dataset into training-sets, validation, and testing sets based on specified percentages. First, it calculates the ratio for the combined test and validation set (`initial_split`) as the sum of the test and validation split percentages, and the ratio for splitting the integrated test and validation set into separate test and validation sets (`test_val_split`). Then, the data (`total_data`) and labels (`total_labels`) are split into training data and combined test/validation data using the `initial_split` ratio. Next, the combined test/validation data is further divided into separate tests and validation sets using the `test_val_split` ratio. The shapes of the resulting training, validation, and testing datasets are printed to verify their dimensions and the successful completion of the data splitting process. Additionally, matplotlib is utilized to contemplate the images from the training-sets, validation, and testing datasets.

```
In [6]: 1 initial_split = test_split_percent*validation_split_percent
2 test_val_split = test_split_percent/initial_split
3
4 train_data, test_val_data, train_labels, test_val_labels = train_test_split(total_data,total_labels,
5                                     test_size=initial_split)
6
7 test_data, val_data, test_labels, val_labels = train_test_split(test_val_data,test_val_labels,
8                                     test_size=test_val_split)
9
10 print('train: ',train_data.shape)
11 print('validation',val_data.shape)
12 print('test',test_data.shape)

train: (4417, 176, 208, 3)
validation (632, 176, 208, 3)
test (1262, 176, 208, 3)
```

Figure 10- Split Data into Train, Validate, Test Sets

To shape method is used to print the dimensions and structure of the training, validation, and test datasets, along with their respective labels.

```
In [7]: 1 print(train_data.shape)
2 print(train_labels.shape)
3 print(val_data.shape)
4 print(val_labels.shape)
5 print(test_data.shape)
6 print(test_labels.shape)

(4417, 176, 208, 3)
(4417, 4)
(632, 176, 208, 3)
(632, 4)
(1262, 176, 208, 3)
(1262, 4)
```

Figure 11- Data and Labels Shapes Output

Several images are displayed for analysis to visualize the appearance of MRI scans.

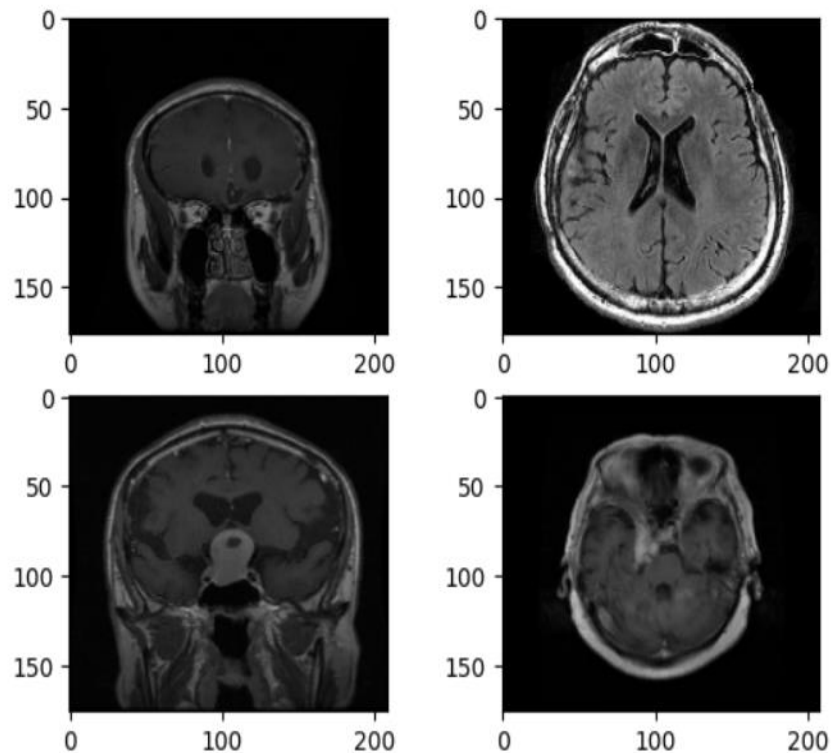


Figure 12- Displaying Sample Images from Train, Validation, and Test Sets

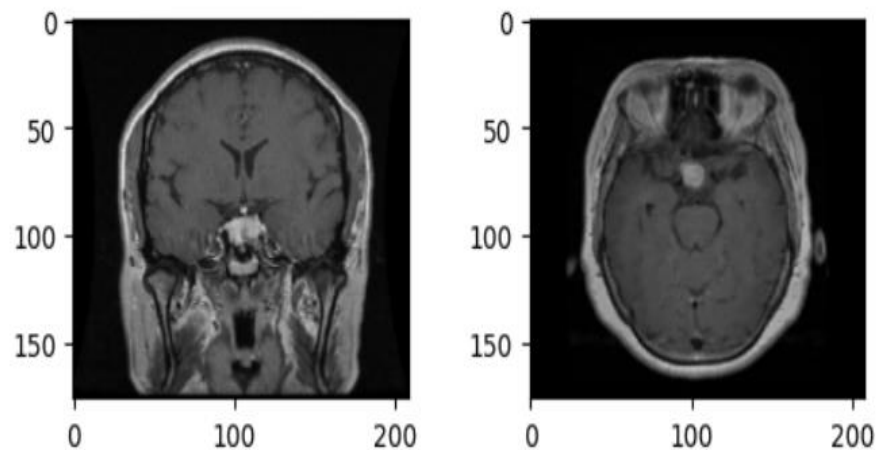


Figure 13- Displaying Sample Images from Train, Validation, and Test Sets

NumPy offers functions named `np.amax` and `np.amin` to determine the maximum and minimum pixel values in the training data and validation data, respectively. The `np.amax` function computes and displays the highest pixel value found among all images in the training dataset, representing the peak intensity or brightness present in any pixel across the training data. Conversely, `np.amin` calculates and prints the lowest pixel value across all images in the training dataset, indicating the minimum intensity or brightness detected in any pixel within the training data. Similarly, these functions are applicable to the validation data to regulate both the minimum and maximum pixel values. These computations yield insights into the range of pixel values present within the datasets, aiding in the evaluation of data normalization, scaling, and the detection of potential anomalies within the data.

```
In [9]: 1 print(np.amax(train_data))
        2 print(np.amin(train_data))
        3 print(np.amax(val_data))
        4 print(np.amin(val_data))

1.0
0.0
1.0
0.0
```

Figure 14- Displaying Maximum and Minimum Pixel Values

A series of subplots are created using Matplotlib's `subplot` function. Four subplots are arranged horizontally in a single row. The first three subplots display individual channels of a specific MRI scan from the training data array, each representing different image channels (assuming RGB channels for visualization). The fourth subplot displays

the entire MRI scan with all channels combined. Each subplot uses the `imshow` function to display the corresponding image data. This visualization helps in understanding the structure and content of MRI scans, particularly how different image channels contribute to the overall image.

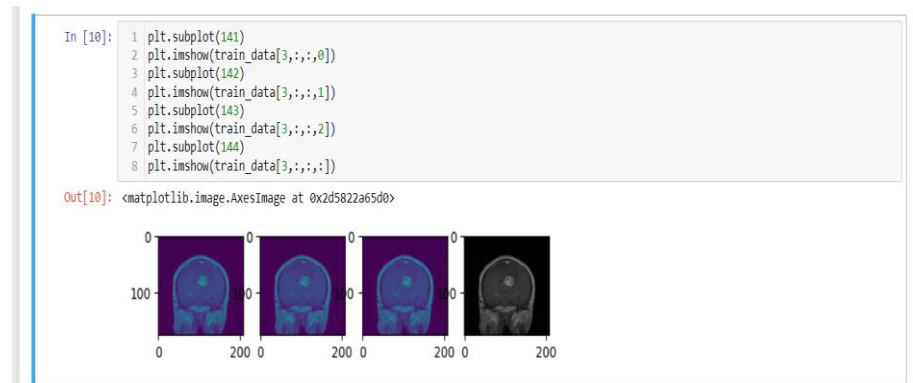


Figure 15- Displaying Separate Channels and Combined Image

The "conv_block" function in TensorFlow and Keras defines and returns a convolutional block as part of a CNN architecture, specifically designed for processing image data. It accepts a single parameter, "filters," which determines the number of filters for the convolutional layers within the block. This function constructs a `tf.keras.Sequential` block, which organizes multiple layers into a sequence to be applied sequentially to the input data. The initial layers consist of `tf.keras.layers.SeparableConv2D`, performing separable convolutions with a 3x3 kernel size and ReLU activation function. Padding is set to "same" to preserve the spatial dimensions of the input data. Following the convolutional layers, a normalization layer is added to stabilize and expedite the training process through batch normalization. Subsequently, a downsampling, which is pooling layer reduces the spatial dimensions of the input while retaining crucial features. Finally, the function returns the `Sequential` block containing the defined sequence of layers, serving as a fundamental building block within the CNN architecture. Overall, this function facilitates the creation of a convolutional block suitable for tasks like image classification.

```

In [11]: 1 def conv_block(filters):
          2     block = tf.keras.Sequential([
          3         tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),
          4         tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),
          5         tf.keras.layers.BatchNormalization(),
          6         tf.keras.layers.MaxPool2D()
          7     ])
          8
          9     return block

```

Figure 16- Function of a Convolutional Block

The "dense_block" function, employing TensorFlow and Keras, defines and returns a dense block for processing data within a neural network. It takes two parameters: "units," indicating the quantity of neurons within the completely related layer, and "dropout_rate," specifying the fraction of devices to drop all through training to prevent overfitting. This function constructs a keras Sequential block, combining layers into a sequence for sequential application to the input data. The initial layer is a dense layer with the specified number of neurons and ReLU activation. Next, a batch normalization layer normalizes the dense layer's output, stabilizing and expediting training. Finally, a dropout layer is added, applying dropout to the data based on the specified dropout rate. Dropout facilitates save you overfitting with the aid of using randomly deactivating enter devices while training. Overall, this function facilitates the creation of a dense block suitable for building neural network architectures for classification or regression tasks.

```

In [12]: 1 def dense_block(units, dropout_rate):
          2     block = tf.keras.Sequential([
          3         tf.keras.layers.Dense(units, activation='relu'),
          4         tf.keras.layers.BatchNormalization(),
          5         tf.keras.layers.Dropout(dropout_rate)
          6     ])
          7
          8     return block

```

Figure 17- Defining of a Dense Block

The "build_model" function constructs and returns a neural network model using TensorFlow and Keras, specifically designed for image classification tasks. It encompasses various kinds of layers along with convolutional, pooling, dropout, dense blocks, and an output layer. The model is structured using tf.keras.Sequential, enabling a sequential stacking of layers. The initial layer serves as an input layer, defining the input shape as a tuple representing an image with specified dimensions and three-color channels (RGB). Subsequently, two convolutional layers with 16 filters and a 3x3 kernel size are added, utilizing the ReLU

activation function and same padding to maintain input size. A MaxPool2D layer follows for downsampling. Three convolutional blocks with filter sizes of 32, 64, and 128 are sequentially incorporated. These blocks consist of separable convolutional layers, batch normalization, and max pooling. Additionally, dropout layers with a dropout rate of 0.2 every are blanketed to save you overfitting. Another convolutional block with 256 filters is introduced, followed by another dropout layer. A flattened layer is then added to transform the input into a 1D array for the subsequent dense layers. Three dense blocks with 512, 128, and 64 units are sequentially appended, each with varying dropout rates of 0.7, 0.5, and 0.3, respectively. These blocks comprise dense layers, batch normalization, and dropout. Finally, the output layer consists of four output units with softmax activation, suitable for a 4-class classification task. In summary, this model is tailored for image classification, utilizing a combination of layers and blocks for features extraction, downsampling, and classification.

```
In [13]: 1 def build_model():
2         model = tf.keras.Sequential([
3             tf.keras.Input(shape=(IMAGE_SIZE, 3)),
4
5             tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
6             tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
7             tf.keras.layers.MaxPool2D(),
8
9             conv_block(32),
10            conv_block(64),
11
12            conv_block(128),
13            tf.keras.layers.Dropout(0.2),
14
15            conv_block(256),
16            tf.keras.layers.Dropout(0.2),
17
18            tf.keras.layers.Flatten(),
19            dense_block(512, 0.7),
20            dense_block(128, 0.5),
21            dense_block(64, 0.3),
22
23            tf.keras.layers.Dense(4, activation='softmax')
24        ])
25
26        return model
```

Figure 18- Model Building Function

The neural network model in TensorFlow and Keras is created and compiled within the scope of a distributed strategy. This ensures parallel training across multiple devices for enhanced performance. The build_model function is invoked within the strategy's scope to initialize the model with distributed compatibility. Performance metrics for training and evaluation are defined, including Binary Accuracy, Precision, Recall, and AUC (Area Under Curve) to assess model performance comprehensively. During compilation, three parameters

are specified: Optimizer, Loss Function, and Metrics. The Adam optimizer, recognized for its efficiency in gradient descent optimization, was chosen. CategoricalCrossentropy loss function is utilized for multi-magnificence classification tasks where outputs are represented as one-hot encoded vectors. Additionally, performance metrics are selected to monitor model performance during training and evaluation. In summary, this setup establishes the neural network model within a distributed training context, specifying the optimizer, loss function, and performance metrics for model compilation. This configuration facilitates effective training across multiple devices while tracking essential metrics for model evaluation.

```
In [14]: 1 with strategy.scope():
2         model = build_model()
3
4         METRICS = [tf.keras.metrics.BinaryAccuracy(name='accuracy'),tf.keras.metrics.Precision(name='precision'),
5                    tf.keras.metrics.Recall(name='recall'),tf.keras.metrics.AUC(name='auc')]
6
7
8         model.compile(
9             optimizer='adam',
10            loss=tf.keras.losses.CategoricalCrossentropy(),
11            metrics=METRICS
12        )
```

Figure 19- Model Compilation and Metric Definitions

The exponential_decay function defines a learning rate scheduler using exponential decay and configures an early stopping mechanism for training a neural network model. It takes parameters such as lr0 for the initial learning rate and S for the step or epoch interval over which the learning rate decays. Within the function, an inner function calculates the new learning rate for each epoch using the formula $lr = lr0 \times 0.1^{\frac{epoch}{s}}$. This formula gradually reduces the learning rate through an element of 0.1 each S epochs, stabilizing the training process as it progresses. The Learning Rate Scheduler callback applies the exponential_decay function at each epoch, vigorously adjusting the learning rate build on the defined schedule. This allows for fine-tuning the learning rate throughout training. Additionally, the Early Stopping Callback is utilized to halt training when a monitored metric stops improving. It continues training for an extra 10 epochs after the last improvement in the monitored metric and then stops if no further improvement is observed. Setting restore_best_weights to True

guarantees that the model's weights revert to the ones accomplishing the very best metric value, aiding in retrieving the model's peak performance even if overfitting occurs later in training. This combination of learning rate scheduling and early stopping enhances the training process, preventing overfitting and ensuring optimal learning rates for faster convergence without overshooting during later learning stages. Such practices are common in training deep learning models to promote robustness and efficiency.

```
In [15]: 1 def exponential_decay(lr0, s):
2         def exponential_decay_fn(epoch):
3             return lr0 * 0.1 ** (epoch / s)
4         return exponential_decay_fn
5
6 exponential_decay_fn = exponential_decay(0.01, 20)
7
8 lr_scheduler = tf.keras.callbacks.LearningRateScheduler(exponential_decay_fn)
9
10 early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)
```

Figure 20- Learning Rate Scheduler and Early Stopping Callbacks

We initiate the model training using the fit method, which utilizes the train_data and train_labels for training while validating with val_data and val_labels. Specifications such as the number of training epochs and batch size are set, determining the number of samples per gradient update. To promote effective learning, shuffle is set to True, ensuring that the training data is shamble at the beginning of each epoch. This prevents the model from memorizing the data order and facilitates better generalization. The training process generates a cnn_model_history object containing comprehensive information on performance metrics like loss and accuracy for each epoch. This permits distinct evaluation of the model's education and validation overall performance at some point of the training process.

```
In [17]: 1 cnn_model_history = model.fit(train_data, train_labels, validation_data=(val_data, val_labels),
2                                     epochs=EPOCHS, batch_size=batch, shuffle=True)
```

Figure 21- Model Training with Training and Validation Data

Pred_labels are employed to generate predictions from the model based on the given test_data. This crucial step entails the model evaluating the provided input data and producing predicted labels. It represents the final stage of the inference process, allowing you to evaluate the model's overall performance on unseen statistics and gauge its capacity to generalize past the training statistics.

```
In [18]: 1 pred_labels = model.predict(test_data)
40/40 [=====] - 6s 132ms/step
```

Figure 22- Making Predictions on Test Data

```
In [19]: 1 pred_labels
Out[19]: array([[8.9786525e-05, 7.0249283e-04, 1.2946500e-05, 9.9919480e-01],
 [1.7894488e-03, 9.9782419e-01, 7.0520939e-05, 3.1582397e-04],
 [1.3392799e-05, 1.4803839e-03, 9.9846894e-01, 3.7158712e-05],
 ...,
 [8.1576931e-05, 6.3283644e-03, 9.9293375e-01, 6.5631478e-04],
 [9.9857974e-01, 7.3068106e-04, 6.7699346e-04, 1.2582033e-05],
 [4.7536207e-05, 4.2533793e-04, 1.3437048e-06, 9.9952579e-01]],
 dtype=float32)
```

Figure 23- Predicted Labels

The model's performance metrics, including accuracy, are calculated using the `model.evaluate` method on `test_data` and `test_labels`. These metrics, including accuracy, are stored in the variable `cnn_accuracy`. The second element in the list specifically denotes the accuracy, which is then printed as a percentage to offer a clear indication of the model's classification performance on the test data.

```
In [20]: 1 cnn_accuracy = model.evaluate(test_data, test_labels)
2 print(f'CNN accuracy: {cnn_accuracy[1]*100}%')
40/40 [=====] - 5s 133ms/step - loss: 0.1995 - accuracy: 0.9719 - precision: 0.9459 - recall: 0.9414 -
auc: 0.9890
CNN accuracy: 97.18700647354126%
```

Figure 24- Evaluation of CNN Model Accuracy

The `summary` function prints a complete review of the neural network model. It displays all the layers within the model, including their shapes and the number of parameters they possess, both trainable and non-trainable. This outline is invaluable for comprehending the model's infrastructure, understanding data flow, and aiding in debugging, particularly to ensure alignment of input and output dimensions across various layers. Additionally, the function presents a total parameter count, assisting in assessing the model's complexity and memory demands.

In [21]: 1 model.summary()		
Model: "sequential_7"		
Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 176, 208, 16)	448
conv2d_1 (Conv2D)	(None, 176, 208, 16)	2320
max_pooling2d (MaxPooling2D)	(None, 88, 104, 16)	0
sequential (Sequential)	(None, 44, 52, 32)	2160
sequential_1 (Sequential)	(None, 22, 26, 64)	7392
sequential_2 (Sequential)	(None, 11, 13, 128)	27072
dropout (Dropout)	(None, 11, 13, 128)	0
sequential_3 (Sequential)	(None, 5, 6, 256)	103296
dropout_1 (Dropout)	(None, 5, 6, 256)	0
Flatten (Flatten)	(None, 7680)	0
sequential_4 (Sequential)	(None, 512)	3934720
sequential_5 (Sequential)	(None, 128)	66176
sequential_6 (Sequential)	(None, 64)	8512
dense_3 (Dense)	(None, 4)	260
=====		
Total params: 4152356 (15.84 MB)		
Trainable params: 4149988 (15.83 MB)		
Non-trainable params: 2368 (9.25 KB)		

Figure 25- Summary of the CNN Model Architecture

The Train_Val_Plot function is designed to plot the training and validation history of a CNN model across numerous metrics consisting of accuracy, loss, AUC, and precision. It accepts the historical data from cnn_model_history, which comprises recorded values of each metric during training. This visualization aids in comprehending the evolution of the model's performance with each epoch, facilitating hyperparameter tuning and diagnosing training issues like overfitting or underfitting. By visualizing how metrics change over time, it becomes easier to assess the effectiveness of training strategies and make informed adjustments to improve model performance.

```

In [22]: 1 def Train_Val_Plot(acc, val_acc, loss, val_loss, auc, val_auc, precision, val_precision):
2         fig, (ax1,ax2,ax3,ax4) = plt.subplots(4,1, figsize=(10,30))
3         fig.suptitle(" MODEL'S METRICS VISUALIZATION of CNN")
4
5         ax1.plot(range(1, len(acc) + 1), acc)
6         ax1.plot(range(1, len(val_acc) + 1), val_acc)
7         ax1.set_title('History of Accuracy')
8         ax1.set_xlabel('Epochs')
9         ax1.set_ylabel('Accuracy')
10        ax1.legend(['training', 'validation'])
11
12        ax2.plot(range(1, len(loss) + 1), loss)
13        ax2.plot(range(1, len(val_loss) + 1), val_loss)
14        ax2.set_title('History of Loss')
15        ax2.set_xlabel('Epochs')
16        ax2.set_ylabel('Loss')
17        ax2.legend(['training', 'validation'])
18
19        ax3.plot(range(1, len(auc) + 1), auc)
20        ax3.plot(range(1, len(val_auc) + 1), val_auc)
21        ax3.set_title('History of AUC')
22        ax3.set_xlabel('Epochs')
23        ax3.set_ylabel('AUC')
24        ax3.legend(['training', 'validation'])
25
26        ax4.plot(range(1, len(precision) + 1), precision)
27        ax4.plot(range(1, len(val_precision) + 1), val_precision)
28        ax4.set_title('History of Precision')
29        ax4.set_xlabel('Epochs')
30        ax4.set_ylabel('Precision')
31        ax4.legend(['training', 'validation'])
32        plt.show()
33        Train_Val_Plot(cnn_model_history.history['accuracy'],cnn_model_history.history['val_accuracy'],
34                        cnn_model_history.history['loss'],cnn_model_history.history['val_loss'],
35                        cnn_model_history.history['auc'],cnn_model_history.history['val_auc'],
36                        cnn_model_history.history['precision'],cnn_model_history.history['val_precision'])

```

Figure 26- Visualization of Training and Validation Metrics

The roundoff function is utilized to convert the predicted probabilities into a one-hot encoded format, wherein the class with the highest probability is imposed a value of 1 while the remaining classes are imposed a value of 0. Subsequently, the classification_report function from the sklearn.metrics module is employed to create a complete report showcasing category metrics together with precision, recall, and F1-rating for every class. This report is based on the comparison between the true labels and the predicted labels. The target_names parameter leverages the classes array to provide meaningful class names within the report, aiding in the interpretation of the results.

```

In [24]: 1 CLASSES = ['Pituitary', 'Notumor', 'Meningioma', 'Glioma']
2         def roundoff(arr):
3             arr[np.argmax(arr) != arr.max()] = 0
4             arr[np.argmax(arr) == arr.max()] = 1
5             return arr
6
7         for labels in pred_labels:
8             labels = roundoff(labels)
9
10        print(classification_report(test_labels, pred_labels, target_names=CLASSES))

```

	precision	recall	f1-score	support
Pituitary	0.97	0.88	0.93	287
Notumor	0.83	0.98	0.90	283
Meningioma	0.98	0.98	0.98	367
Glioma	1.00	0.92	0.96	325
micro avg	0.94	0.94	0.94	1262
macro avg	0.95	0.94	0.94	1262
weighted avg	0.95	0.94	0.94	1262
samples avg	0.94	0.94	0.94	1262

Figure 27- Classification Report for Model Evaluation

A graph displaying the training and validation accuracy of the CNN model across different epochs. The x-axis represents the epochs, even as the y-axis represents the accuracy. The stable blue line shows the training accuracy, and the stable yellow line represents the validation accuracy.

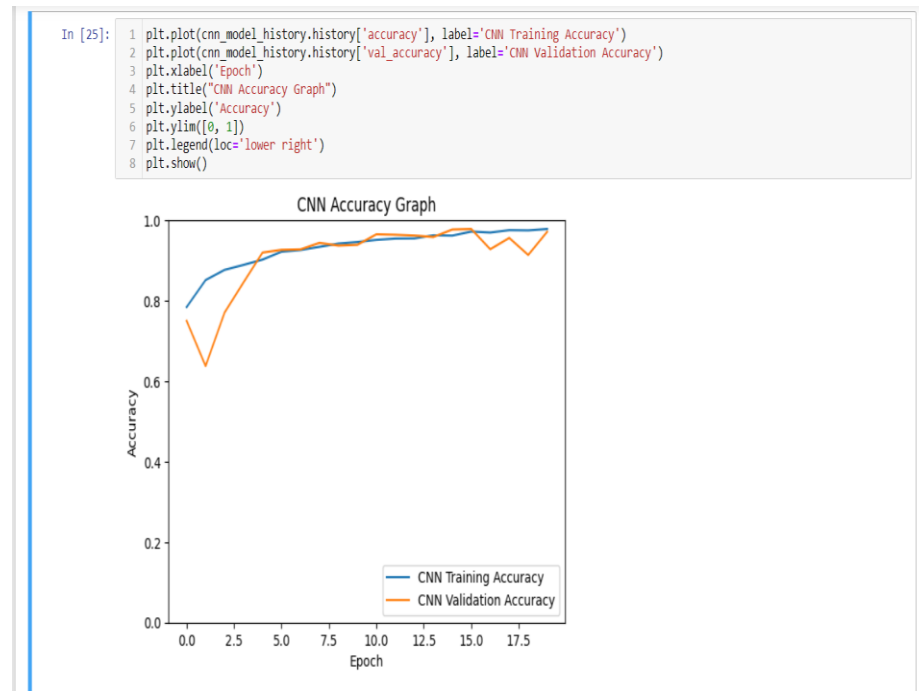


Figure 28- Plotting Training and Validation Accuracy

For reshaping the training and testing data arrays to prepare them for input into an RNN, the reshape function is employed. This function adjusts the shape of an array while retaining the same number of elements. In this scenario, train_data and test_data arrays are reshaped. train_data.shape[0] and test_data.shape[0] denote the number of samples in the training and testing data, respectively, while train_data.shape[1] and test_data.shape[1] represent the height of each image. The parameter -1 indicates that the length of each image is flattened during reshaping. This reshaping operation is necessary because RNNs typically anticipate input data in a specific format.

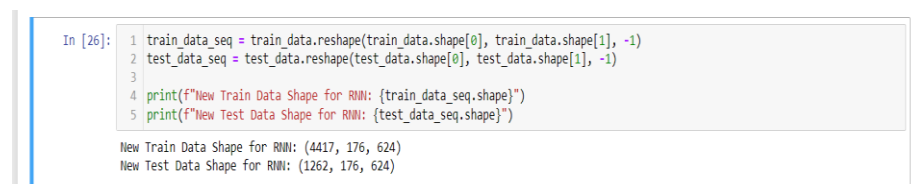


Figure 29- Reshaping Data for RNN

In the RNN model architecture, we import necessary modules from TensorFlow Keras to build and compile the RNN model. Two SimpleRNN layers are added, each specifying the number of units as 50 and 20 respectively, along with the activation function. The input shape is determined by the input shape parameter. The first SimpleRNN layer has return_sequences set to True, indicating that it returns the full sequence of outputs rather than just the last output. Following the SimpleRNN layers, two Dense layers are added. The first Dense layer contains sixty-four devices with relu activation, whilst the second one Dense layer has four devices with softmax activation for multi-magnificence classification. The input_shape variable is set to the shape of the input data, considering index numbers 1 and 2, corresponding to the height and flattened length of each image in the training data sequence. The model is compiled with the use of the Adam optimizer, specific cross-entropy loss function, and accuracy because of the evaluation metric. Finally, the model summary technique is invoked to show a precis of the version architecture, which include layer types, output shapes, and the quantity of parameters.

```
In [27]: 1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import SimpleRNN, Dense
3
4 def build_rnn_model(input_shape):
5     model = Sequential([
6         SimpleRNN(50, activation='relu', input_shape=input_shape, return_sequences=True),
7         SimpleRNN(20, activation='relu'),
8         Dense(64, activation='relu'),
9         Dense(4, activation='softmax')
10    ])
11
12    return model
13
14 input_shape = (train_data_seq.shape[1], train_data_seq.shape[2])
15
16 model = build_rnn_model(input_shape)
17 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
18 model.summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 176, 50)	33750
simple_rnn_1 (SimpleRNN)	(None, 20)	1420
dense_4 (Dense)	(None, 64)	1344
dense_5 (Dense)	(None, 4)	260

=====
Total params: 36774 (143.65 KB)
Trainable params: 36774 (143.65 KB)
Non-trainable params: 0 (0.00 Byte)

Figure 30- Building an RNN Model

The model.fit() approach is invoked to train the RNN model. It accepts arguments such as the training data sequence, training labels, epoch

value, validation data, and the batch size. The training data sequence is a 3D array containing input data formatted for RNN processing. Training labels correspond to the training data, providing the ground truth for model learning. Epochs are set to specify the number of training iterations. Validation data sequence and labels are used to monitor the model's overall performance on unseen data during training, aiding in assessing generalization ability. The batch size parameter determines the wide variety of samples processed earlier than updating the model's parameters, influencing training efficiency and memory usage. Once training is complete, the evaluation method is employed to evaluate the model's overall performance on the test data sequence and labels. It returns metrics such as loss and accuracy, providing insights into the model's effectiveness on unseen test data.

```
In [28]: 1 rnn_history = model.fit(train_data_seq, train_labels, epochs=35,  
2                               validation_data=(test_data_seq, test_labels),  
3                               batch_size=32)  
4 loss, accuracy = model.evaluate(test_data_seq, test_labels)  
5 print(f"Test Accuracy: {accuracy*100:.2f}%")
```

Figure 31- Training and Evaluating RNN Model

A graph showing the training and validation accuracy of the RNN version throughout one-of-a-kind epochs. The x-axis represents the epochs, even as the y-axis represents the accuracy. The stable blue line suggests the training accuracy, and the stable yellow line represents the validation accuracy.

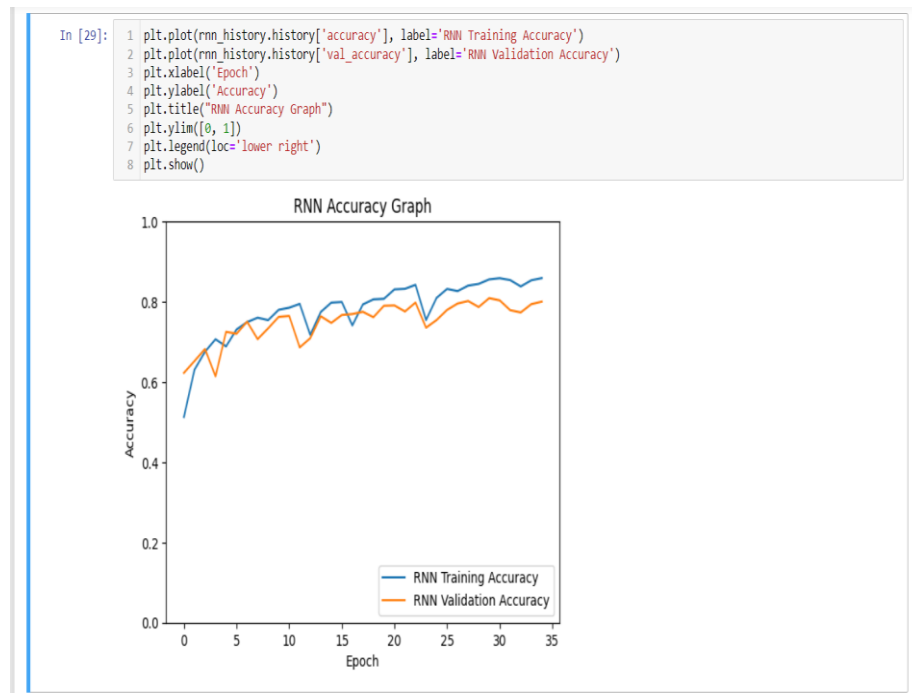


Figure 32- Plotting Training and Validation Accuracy for RNN Model

The OASIS dataset classifies individuals into four distinct categories: non-demented, very mild demented, mild demented, and moderate demented. Non-demented individuals exhibit cognitive functioning typical for their age, devoid of any signs of dementia. Conversely, the very mild demented stage marks the initial phase of cognitive decline, often overlooked due to its subtle nature, where minor memory lapses may occur. As symptoms progress to the mild demented stage, cognitive challenges intensify, impacting complex tasks and memory retention, necessitating lifestyle adjustments and increased assistance. Finally, the moderate demented stage witnesses a significant deterioration in cognitive abilities, manifesting in profound difficulties with daily activities, heightened memory loss, and impaired language and reasoning skills, requiring extensive support and supervision for basic tasks. These stages illustrate the evolving nature of cognitive impairment, underscoring the importance of tailored care and intervention strategies to enhance quality of life.

Importing essential libraries for building and training neural network models for image classification tasks. It imports fundamental libraries for deep learning, image manipulation, and data preprocessing. Furthermore, it covers the inclusion of tools essential for dividing data

into training and test sets, as well as for assessing and displaying the performance of the models.

```
In [1]: 1 import keras
2 import os
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7 from keras.models import Sequential
8 import PIL
9 from PIL import Image
10 from keras.layers import Conv2D, Flatten, Dense, Dropout, BatchNormalization, MaxPooling2D
11 from sklearn.preprocessing import OneHotEncoder
12 from sklearn.model_selection import train_test_split
```

Figure 33- Importing Necessary Libraries

Initially, we create four empty lists named `non_demented`, `very_mild_demented`, `mild_demented`, and `moderate_demented`. We then proceed by navigating through the directory structure of designated folders, each containing image data representing various stages of dementia. Through this traversal, we compile the file paths of images linked to each dementia stage, thus setting the stage for efficient data handling and preparation for model training activities.

```
In [2]: 1 non_demented = []
2 very_mild_demented = []
3 mild_demented = []
4 moderate_demented = []
5
6 for dirname, _, filenames in os.walk('./OASIS_DATA/Non_Demented'):
7     for filename in filenames:
8         non_demented.append(os.path.join(dirname, filename))
9
10 for dirname, _, filenames in os.walk('./OASIS_DATA/Very_mild_Dementia'):
11     for filename in filenames:
12         very_mild_demented.append(os.path.join(dirname, filename))
13
14 for dirname, _, filenames in os.walk('./OASIS_DATA/Mild_Dementia'):
15     for filename in filenames:
16         mild_demented.append(os.path.join(dirname, filename))
17
18 for dirname, _, filenames in os.walk('./OASIS_DATA/Moderate_Dementia'):
19     for filename in filenames:
20         moderate_demented.append(os.path.join(dirname, filename))
```

Figure 34-Gathering Image File Paths for Different Classes

Some of the images are displayed using the matplotlib and PIL libraries.

```
In [46]: 1 fig, axes = plt.subplots(2, 2, figsize=(20, 10))
2
3 axes[0, 0].imshow(PIL.Image.open(str(non_demented[10])))
4 axes[0, 0].set_title('Non-Demented')
5
6 axes[0, 1].imshow(PIL.Image.open(str(very_mild_demented[20])))
7 axes[0, 1].set_title('Very Mild Demented')
8
9 axes[1, 0].imshow(PIL.Image.open(str(mild_demented[44])))
10 axes[1, 0].set_title('Mild Demented')
11
12 axes[1, 1].imshow(PIL.Image.open(str(moderate_demented[99])))
13 axes[1, 1].set_title('Moderate Demented')
14
15 plt.show()
```

Figure 35- Displaying Sample Images from Each Class

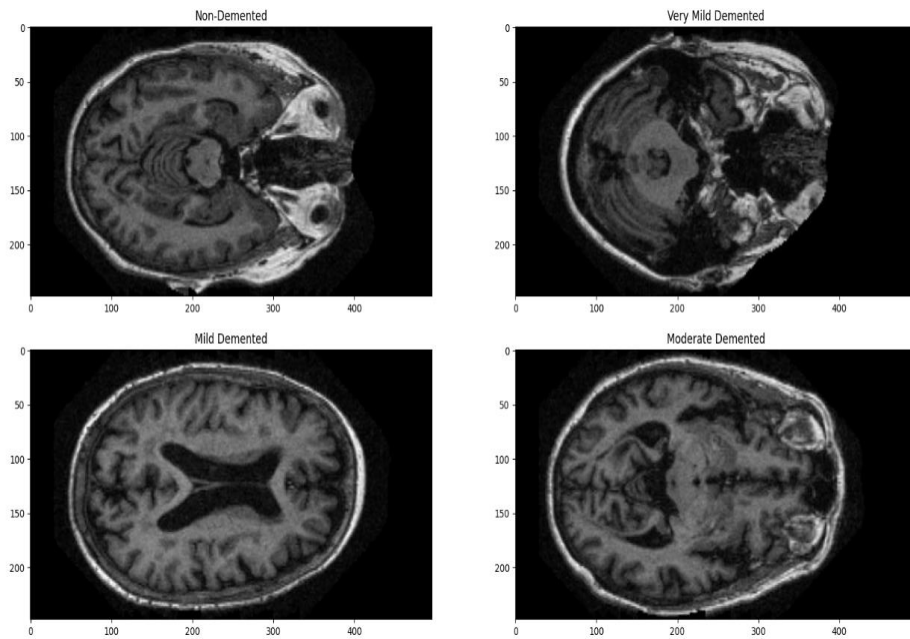


Figure 36- Images from Each Class

Prints the count of images gathered for each stage of dementia.

```
In [4]: 1 print(len(non_demented))
        2 print(len(mild_demented))
        3 print(len(moderate_demented))
        4 print(len(very_mild_demented))

67222
4997
488
13725
```

Figure 37- Printing the Number of Images in Each Class

Modifying all the lists into the same number of images. This helps to balance the dataset or for computational efficiency during subsequent data processing and model training stages.

```
In [5]: 1 non_demented=non_demented[0:488]
        2 mild_demented=mild_demented[0:488]
        3 very_mild_demented=very_mild_demented[0:488]
```

Figure 38- Trimming the Number of Images in Each Class to Balance the Dataset

```
In [6]: 1 print(len(non_demented))
        2 print(len(mild_demented))
        3 print(len(moderate_demented))
        4 print(len(very_mild_demented))

488
488
488
488
```

Figure 39- Printing the Updated Number of Images in Each Class

One-hot encoding, a process commonly used to convert categorical variables into a numerical format suitable for machine learning algorithms. The encoders learn the mapping between each category and

its corresponding numerical representation, enabling subsequent transformation of categorical labels into one-hot encoded vectors.

```
In [7]: 1 # One Hot Encoding
        2 # 0 -> non_demented
        3 # 1 -> mild_dementia
        4 # 2 -> moderate_dementia
        5 # 3 -> very_mild_dementia
        6 encoder= OneHotEncoder()
        7 encoder.fit([[0],[1],[2],[3]])

Out[7]: OneHotEncoder()
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

Figure 40- Performing One Hot Encoding

This describes the process of preparing image data from different dementia stages for machine learning tasks. It involves iterating through stored file paths of images, loading each one, resizing it to a uniform dimension of 128x128 pixels, and converting it to a NumPy array format. The images are only appended to the dataset if they match the desired shape of (128, 128, 3), ensuring consistency. Alongside this, the dementia stage of each image is encoded using one-hot encoding, which simplifies handling categorical data for machine learning models. These encoded labels are then paired with their corresponding images in the dataset. This structured approach efficiently prepares the data for training neural network models by ensuring that both inputs (images) and outputs (labels) are properly formatted and aligned.

```
In [8]: 1 data = []
        2 result = []
        3 for s in non_demented:
        4     img = Image.open(s)
        5     img = img.resize((128,128))
        6     img = np.array(img)
        7     if(img.shape == (128,128,3)):
        8         data.append(np.array(img))
        9         result.append(encoder.transform([[0]]).toarray())
        10
        11 for s in mild_demented:
        12     img = Image.open(s)
        13     img = img.resize((128,128))
        14     img = np.array(img)
        15     if(img.shape == (128,128,3)):
        16         data.append(np.array(img))
        17         result.append(encoder.transform([[1]]).toarray())
        18
        19 for s in moderate_demented:
        20     img = Image.open(s)
        21     img = img.resize((128,128))
        22     img = np.array(img)
        23     if(img.shape == (128,128,3)):
        24         data.append(np.array(img))
        25         result.append(encoder.transform([[2]]).toarray())
        26
        27 for s in very_mild_demented:
        28     img = Image.open(s)
        29     img = img.resize((128,128))
        30     img = np.array(img)
        31     if(img.shape == (128,128,3)):
        32         data.append(np.array(img))
        33         result.append(encoder.transform([[3]]).toarray())
```

Figure 41- Processing Images and Labels for Model Training

The `np.array` function is utilized to transform a list of image data into a NumPy array, effectively creating a structured format suitable for machine learning models. This array serves as the input data, with each element representing an individual image, ready for processing and analysis in subsequent model training steps.

```
In [9]: 1 X=np.array(data)
```

Figure 42- Converting Data to NumPy Array

```
In [10]: 1 X.shape  
Out[10]: (1952, 128, 128, 3)
```

Figure 43- Shape of the Data Array

The code processes the labels for the machine learning model by first converting the list of encoded labels into a NumPy array. This array is then reshaped to match the number of rows in the input data, each row having four columns that correspond to the four dementia categories. To revert the one-hot encoded labels back to their original categorical format, the code applies `np.argmax` along axis 1. This results in an array of integer labels, where each integer corresponds to a specific stage of dementia.

```
In [11]: 1 y=np.array(result)  
         2 y=y.reshape(X.shape[0],4)  
         3 y=np.argmax(y, axis=1)  
         4 y  
Out[11]: array([0, 0, 0, ..., 3, 3, 3], dtype=int64)
```

Figure 44- Converting Labels to NumPy Array

The ``train_test_split`` characteristic is used to divide the dataset into training and trying out subsets. ``X_train`` and ``y_train`` are the functions and labels of the training set, respectively, while ``X_test`` and ``y_test`` constitute the functions and labels of the trying out set. The ``test_size`` parameter determines the fraction of the dataset allotted to the check set, set at 20% on this case. The ``random_state`` parameter is used to make sure that the breakup is reproducible throughout unique executions; putting it to forty-two ensures that the department of facts stays regular on every occasion the code is run.

```
In [12]: 1 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

Figure 45- Splitting Data into Training and Testing Sets

The CNN model is constructed using the Keras Sequential API. It begins with a 2D convolutional layer with 32 filters, a (2,2) kernel size, and 'same' padding to maintain spatial dimensions. The input shape is defined for 128x128 RGB images. Additional convolutional layers with sixty-four and 128 filters are delivered to seize an increasing number of complicated features. Batch normalization aids in stabilizing training, even as max pooling reduces spatial dimensions even as keeping crucial features. Dropout is integrated to mitigate overfitting via way of means of randomly deactivating neurons at some stage in training. The Flatten layer reshapes the output from convolutional layers right into a 1D vector for next related layers. Two dense layers with 256 neurons and ReLU activation are introduced, accompanied via way of means of dropout regularization for similarly generalization. The final layer, with 4 neurons and sigmoid activation, outputs class probabilities for multi-label classification, treating each class independently.

```
In [15]: 1 model = Sequential()
2 model.add(Conv2D(filters=32, kernel_size=2, padding='Same', input_shape = (128,128,3)))
3 model.add(Conv2D(filters=32, kernel_size=2, padding='Same', activation='relu'))
4 model.add(BatchNormalization())
5 model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
6 model.add(Dropout(0.25))
7 model.add(Conv2D(filters=64, kernel_size=2, padding='Same', activation='relu'))
8 model.add(Conv2D(filters=64, kernel_size=2, padding='Same', activation='relu'))
9 model.add(BatchNormalization())
10 model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
11 model.add(Dropout(0.25))
12 model.add(Conv2D(filters=128, kernel_size=2, padding='Same', activation='relu'))
13 model.add(Conv2D(filters=128, kernel_size=2, padding='Same', activation='relu'))
14 model.add(BatchNormalization())
15 model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
16 model.add(Dropout(0.25))
17 model.add(Flatten())
18 model.add(Dense(256, activation='relu'))
19 model.add(Dropout(0.25))
20 model.add(Dense(4, activation='sigmoid'))
```

Figure 46- Creating a CNN Model

The model.summary function provides a concise overview of the defined neural network model. It consists of statistics approximately every layer, consisting of its type, output shape, and the number of trainable parameters. This summary is useful for comprehending the architecture of the model, the flow of data through it, and assessing its complexity and memory requirements.

```
In [16]: 1 print(model.summary())
```

Model: "sequential"		
Layer (type)	Output Shape	Param #

conv2d (Conv2D)	(None, 128, 128, 32)	416
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4128
batch_normalization (Batch Normalization)	(None, 128, 128, 32)	128
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
dropout (Dropout)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	8256
conv2d_3 (Conv2D)	(None, 64, 64, 64)	16448
batch_normalization_1 (Batch Normalization)	(None, 64, 64, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_1 (Dropout)	(None, 32, 32, 64)	0
conv2d_4 (Conv2D)	(None, 32, 32, 128)	32896
conv2d_5 (Conv2D)	(None, 32, 32, 128)	65664
batch_normalization_2 (Batch Normalization)	(None, 32, 32, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 128)	0
dropout_2 (Dropout)	(None, 16, 16, 128)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 256)	838864
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1028

Total params: 8518596 (32.50 MB)		
Trainable params: 8518148 (32.49 MB)		
Non-trainable params: 448 (1.75 KB)		
None		

Figure 47- Summary of the CNN Model Architecture

The `model.compile` function is essential for preparing the model for training. It allows us to specify crucial parameters which includes the optimizer, loss function, and assessment metrics. In this case, the Adam optimizer is chosen for efficient gradient descent optimization. The loss characteristic is about to sparse express crossentropy, that's appropriate for multi-magnificence class tasks. Lastly, accuracy is selected as the assessment metric to evaluate the model's overall performance during training and evaluation phases. This configuration ensures that the model is ready for the subsequent training process.

```
In [17]: 1 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 48- Compiling the CNN Model

The `EarlyStopping` callback in Keras is used to monitor a specified metric during training and stop the training process early if the monitored metric stops improving. It monitors the validation loss and stops training if the change in validation loss is less than `min_delta` for several patience epochs. The `min_delta` parameter defines the threshold for considering an improvement and specifies the number of epochs to wait before stopping. Other parameters allow for customization of the callback's behavior, such as verbosity and restoring best weights.

```
In [18]: 1 from keras.callbacks import EarlyStopping
2 early_stopping=EarlyStopping(
3     monitor='val_loss',
4     min_delta=0.00001,
5     patience=20,
6     verbose=0,
7     mode='auto',
8     baseline=None,
9     restore_best_weights=False,
10    start_from_epoch=0,
11 )
```

Figure 49- Setting up Early Stopping Callback

The `model.fit` method is utilized to train the CNN model using the provided training data, with a validation split of 20% for monitoring model performance. The training process is capped at a maximum of 50 epochs, with early stopping activated to prevent overfitting, set to halt training if no improvement is observed for 20 consecutive epochs. Each training iteration processes batches of 32 samples. The training progress and validation metrics are captured and saved within the `history_cnn` variable for next evaluation and visualization.

```
In [19]: 1 history_cnn= model.fit(X_train,y_train, validation_split=0.2,epochs=50,callbacks=[early_stopping],batch_size=32)
```

Figure 50- Training the CNN Model with Early Stopping Callback

The trained CNN model is evaluated using the test data to assess its performance. The model's accuracy on the test data is computed and printed, providing an indication of how nicely the model generalizes to unseen data.

```
In [20]: 1 cnn_accuracy = model.evaluate(X_test,y_test)
2 print(f'Accuracy of CNN: {cnn_accuracy[1]*100}%')

13/13 [=====] - 2s 138ms/step - loss: 9.0245e-08 - accuracy: 1.0000
Accuracy of CNN: 100.0%
```

Figure 51- Evaluating the CNN Model Accuracy

A line plot depicting the training and validation accuracy over epochs for the previously trained CNN model is generated. This visualization offers insights into the evolution of accuracy during the training process, facilitating the evaluation of the model's performance and identifying any potential overfitting or underfitting issues. The accuracy graph of the CNN model displays consistent results, reaching 100% accuracy.

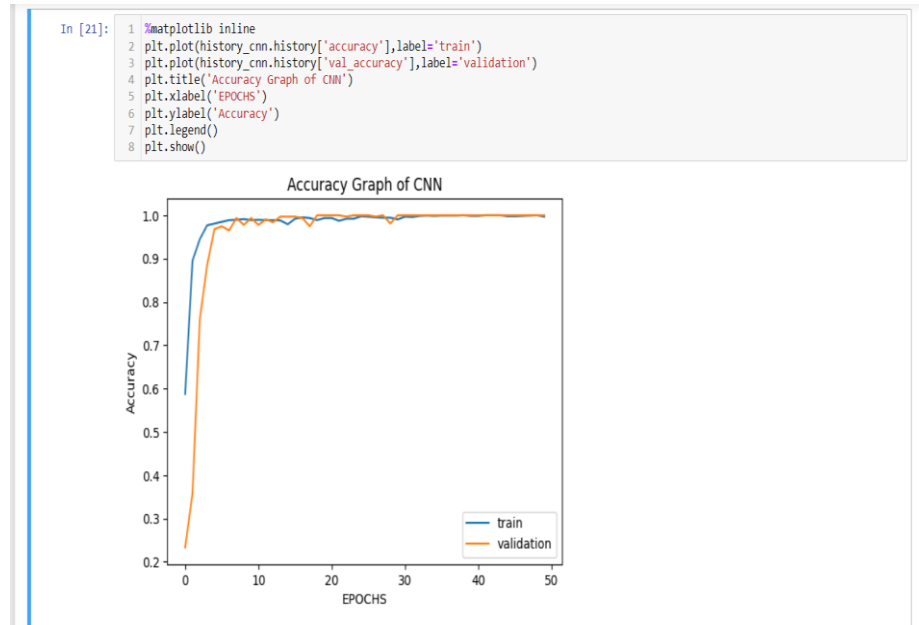


Figure 52- Plotting Training and Validation Accuracy for CNN Model

Using matplotlib the line plot showing the training and validation loss over epochs for the CNN model trained earlier. It provides insights into how the loss changes during the training process, helping to assess the model's performance and potential overfitting and underfitting issues.

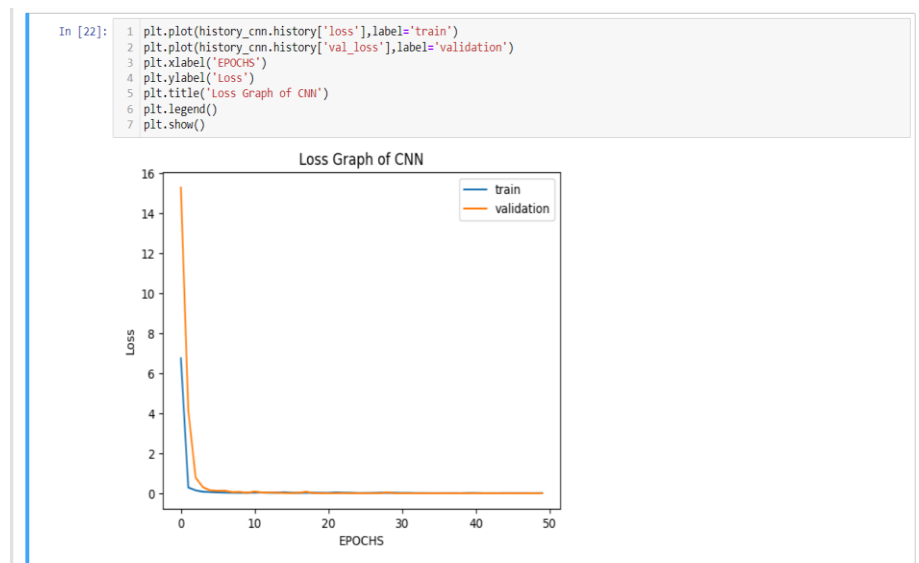


Figure 53- Plotting Training and Validation Loss for CNN Model

To transition from a CNN to an RNN architecture like LSTM, it's common to remove the last layer from the model and then retrieve the output shape of the new last layer. This allows for seamless integration of the model with subsequent layers, ensuring compatibility between the CNN and RNN components.


```
In [23]: 1 from keras.layers import LSTM
```

Figure 54- Importing LSTM Layer

```
In [24]: 1 model.pop()
```

Figure 55- Removing the Last Layer of the Model

```
In [25]: 1 input_shape = model.layers[-1].output_shape[1:]
```

Figure 56- Setting up Input Shape for the LSTM Layer

To adapt the model architecture, we're incorporating layers to smoothly transition from the convolutional layers to LSTM layers. This involves reshaping the output of the preceding layer to match the LSTM input shape, applying TimeDistributed layer to apply the Flatten operation across time steps, integrating LSTM layers with defined units and activation functions, and ultimately adding a Dense layer with softmax activation for classification purposes.

```
In [26]: 1 from keras.layers import Reshape, TimeDistributed
2 model.add(Reshape((1,) + input_shape))
3 model.add(TimeDistributed(Flatten()))
4 model.add(LSTM(units=128, activation='relu', return_sequences=True))
5 model.add(LSTM(units=64, activation='relu'))
6 model.add(Dense(4, activation='softmax'))
```

Figure 57- Adding LSTM Layers to the Model

The model is configured with the Adam optimizer, using sparse specific crossentropy because the loss function, and accuracy because the assessment metric.

```
In [27]: 1 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 58- Compiling the RNN Model

The RNN model is trained on the training data with a validation split of 20%, running for a maximum of 50 epochs. Early stopping is implemented as a callback to prevent overfitting, with a batch size of 32 for training.

```
In [28]: 1 history_rnn = model.fit(X_train, y_train, validation_split=0.2, epochs=50, callbacks=[early_stopping], batch_size=32)
```

Figure 59- Training the RNN Model with Early Stopping Callback

The trained RNN version is evaluated the use of the check data, and the accuracy is printed.

```
In [29]: 1 rnn_accuracy = model.evaluate(X_test, y_test)
2 print(f'accuracy of RNN: {rnn_accuracy[1]*100}%')

13/13 [=====] - 2s 143ms/step - loss: 0.0102 - accuracy: 0.9949
accuracy of RNN: 99.48849081993103%
```

Figure 60- Evaluating the RNN Model Accuracy

Using Matplotlib, visualize the training and validation accuracy of the RNN model across epochs. The accuracy is recorded as 99.48%.

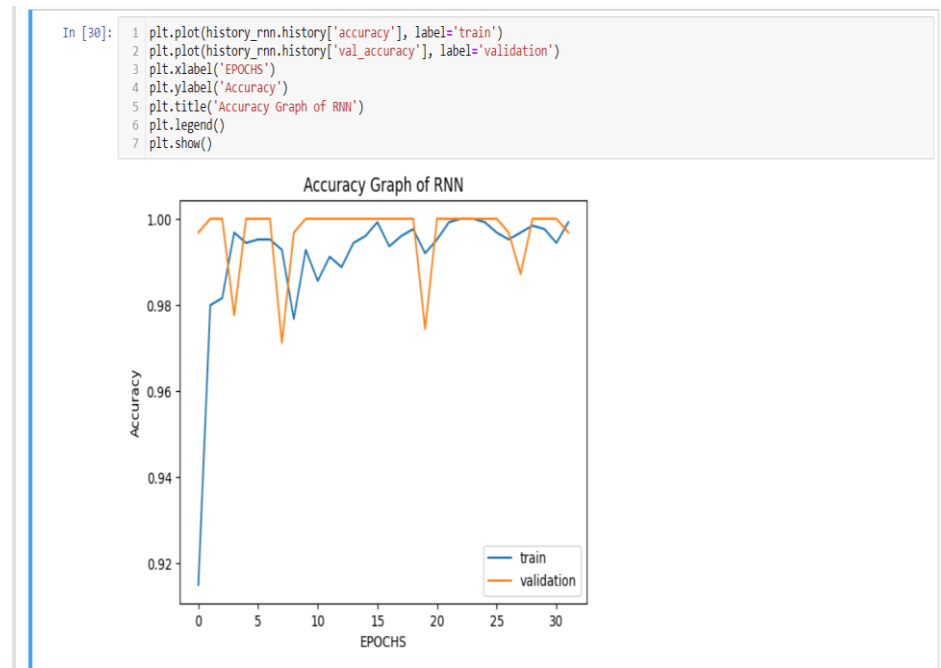


Figure 61- Plotting Training and Validation Accuracy for RNN Model

Matplotlib is utilized to showcase the training and validation loss of the RNN model across epochs in the graph.

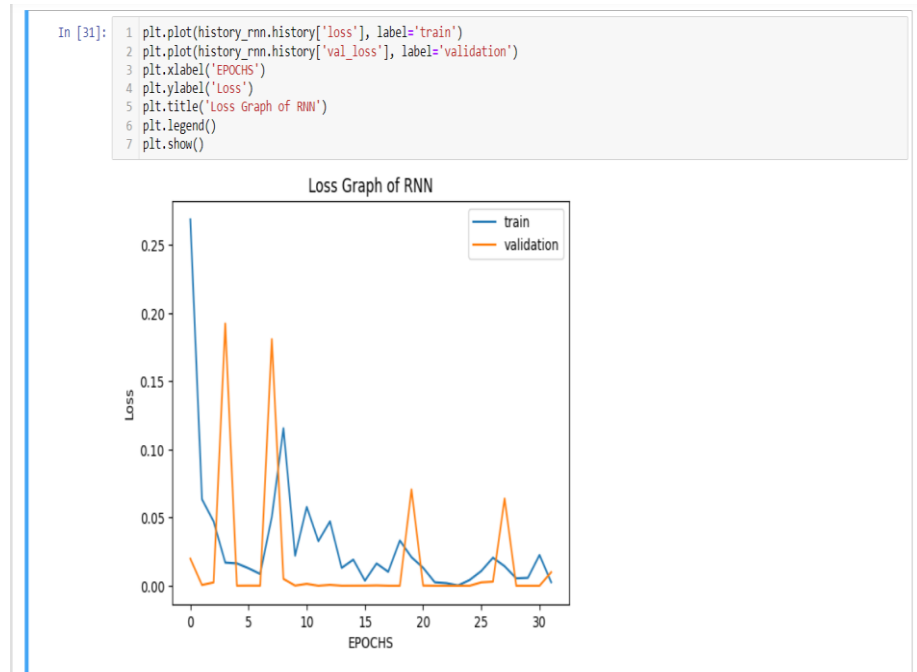


Figure 62- Plotting Training and Validation Loss for RNN Model

Our research reveals the effectiveness of CNNs and RNNs in categorizing brain images into distinct groups. CNNs achieved remarkable accuracies of 100% and 97.33% on the OASIS and Kaggle datasets, respectively, while RNNs achieved slightly lower accuracies of 73.15% and 80.67% on the same datasets. This discrepancy underscores the strengths of each architecture; CNNs excel in capturing spatial information, whereas RNNs excel in capturing temporal dependencies and sequential patterns. Our findings underscore the importance of selecting the appropriate architecture based on task specifics and data characteristics. Looking ahead, advancements in classification accuracy and clinical relevance are anticipated through larger datasets, refined model architectures, and exploration of advanced data augmentation techniques. Overall, our study underscores the capability of deep learning models in accurately classifying brain images and stresses the ongoing need for research and development to address clinical challenges effectively.

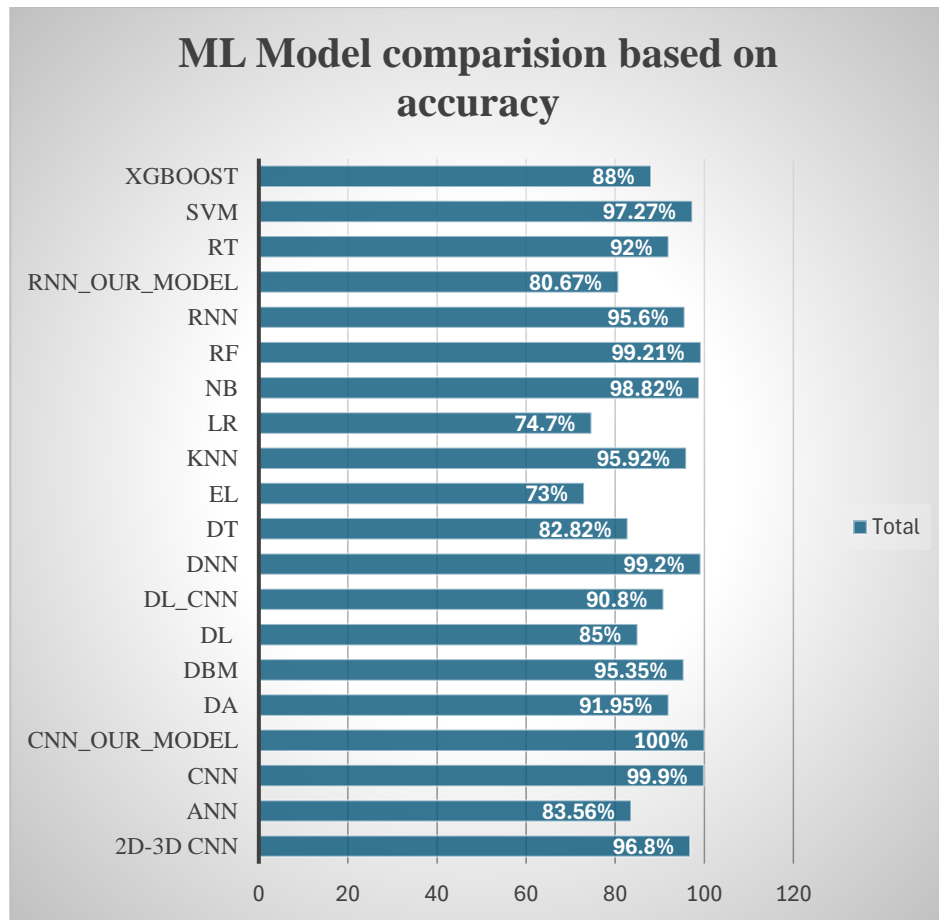


Figure 63- Performance comparison of proposed model with existing models.

Final Chapter: Conclusion

This research delves deep into the transformative potential of CNN and RNN in analysing brain images across a diverse spectrum of populations. It underscores the remarkable capacity of CNNs to apprehend intricate patterns and discern conflicts within these images, demonstrating a clear advancement over previous methodologies. Notably, CNNs achieved extraordinary accuracy rates of 100% and 97.33% on the OASIS and Kaggle datasets, respectively, while RNNs, although slightly trailing, still exhibited commendable accuracy at 73.15% and 80.67%.

These findings underscore the critical importance of selecting neural network architectures that are finely attuned to the specific demands and nuances of each task and dataset. Furthermore, the research points towards promising avenues for future exploration, highlighting the necessity for larger datasets, refined model designs, and cutting-edge data augmentation techniques to further bolster classification accuracy and treatment efficacy.

In the broader context of scientific inquiry, this study offers tantalizing insights that inspire both scientists and clinicians to make more precise diagnoses and treatment decisions. By harnessing the power of CNNs and RNNs to discern intricate patterns within brain images, this research not only showcases the evolution of deep learning but also underscores the collaborative nature of therapeutic research endeavours.

Looking towards the horizon, the study paints a vision of a future characterized by excellence and innovation, fuelled by advances in big data analytics and neural network optimization. By leveraging the transformative potential of these technologies, researchers and clinicians can forge a path towards a healthcare landscape wherein accurate diagnosis and treatment are not mere aspirations but tangible realities.

Expanding on these themes, it becomes apparent that this research holds significant implications for the future of healthcare. The integration of CNNs and RNNs into clinical practice has the potential to revolutionize medical decision-making, enabling practitioners to make more informed

and effective choices for their patients. Furthermore, by leveraging these advanced computational techniques, healthcare providers can enhance their ability to tailor treatments to individual patients, thereby optimizing outcomes and improving patient care.

Moreover, this research underscores the critical role of collaboration in driving scientific progress. By bringing together experts from diverse fields, including computer science, neuroscience, and medicine, researchers can leverage their collective expertise to tackle some of the most pressing challenges in healthcare. This collaborative approach fosters cross-pollination of ideas and facilitates the development of innovative solutions that have the potential to transform the healthcare landscape.

In conclusion, this research serves as a rallying cry for increased investment in efficiency and innovation within the scientific and medical communities. By embracing the transformative power of deep learning and fostering collaborative partnerships, we can unlock new frontiers in precision medicine and usher in a future where accurate diagnosis and treatment are not just aspirations but everyday realities. Through concerted effort and collective action, we can chart a course towards a brighter, healthier future for all.

Publication Details

Title: Exploring Neural Network Approaches for Alzheimer's Disease Detection: An Analysis of RNN and CNN Performance

Conference: International Conference on Communication, Computer Sciences and Engineering (IC3SE-2024) at Amity University, Greater Noida



Fateh Singh Rana <fatehsinghrana130302@gmail.com>

IC3SE-2024 #0931-- Acceptance Letter

1 message

Amity Conference <conference@gn.amity.edu> Wed, May 1, 2024 at 4:12 PM
To: "rs9110131217@gmail.com" <rs9110131217@gmail.com>, Sandeep Kumar <ssihag222@gmail.com>, "fatehsinghrana130302@gmail.com" <fatehsinghrana130302@gmail.com>, "salonisingh1312@gmail.com" <salonisingh1312@gmail.com>, "akarsiharsh12@gmail.com" <akarsiharsh12@gmail.com>, "dikshitkukreja5@gmail.com" <dikshitkukreja5@gmail.com>, "neelsharma087@gmail.com" <neelsharma087@gmail.com>
Cc: Chair Conference <conferencechair@gn.amity.edu>

Dear Author,

Greetings from [Amity University!](#)

We are pleased to inform that your manuscript ID: 0931 with Title: "Exploring Neural Network Approaches for Alzheimer's Disease Detection: An Analysis of RNN and CNN Performance", submitted to 2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE-2024) at Amity University, Greater Noida, has been accepted. Kindly consider this email as formal acceptance of this submission.

The Conference is scheduled to be held from 09th to 11th May 2024.

Registration

Please pay the registration fee for this submission, at the earliest. *Registration fee category must be chosen based on affiliation and membership of 1st author only.* For registration fee submission, please visit [Registration webpage](#), on Conference Website.

If the 1st author of the submission is a valid IEEE Member, you may avail a discount benefit in registration fee as mentioned on Registration Webpage. In case the discount is availed, kindly send us the IEEE Membership Letter/ Card for the same, failing to which, you will be required to submit the availed discounted amount.

Final Version Manuscript

After registration, you are required to submit the final version of the manuscript in [IEEE Template](#) in both .pdf and .docx (MS Word) format, which must include the suggestions/ changes as per instructions/ improvement mentioned by the Technical Program Committee/ Program Committee Members/ Reviewers in the comments, if any. These changes must be limited to 20% part of the manuscript.

<https://mail.google.com/mail/u/0/?ik=39fb0eaf4f&view=pt&search=all&permthid=thread-f:1797846603011858328&simpl=msg-f:1797846603011858328>

1/3

5/2/24, 5:18 PM

Gmail - IC3SE-2024 #0931-- Acceptance Letter

The final version must be submitted to our *Publication Support Wing (PSW)* (publicationsupport@gn.amity.edu). Final version of manuscript must comply with IC3I-2023 [Submission Guidelines](#).

Preprint

Once you will submit the final version of your manuscript to PSW, PSW will check for format of the same as per [IEEE Template](#), will amend if required keeping the content of manuscript unchanged and get it validated/confirmed from you. After validation/ confirmation from your end, PSW will get the preprint ready at its end.

e-Copyright Transfer

After all the above formalities, you will also be receiving an email from IEEE (ECopyright@ieee.org) for e-signing of e-copyright transfer form. Transfer of copyright to IEEE is a mandatory requirement. Our PSW will also intimate you via email, once it will initiate the Copyright Transfer Process. Copyright Transfer will be a self-explanatory process. Still, if any assistance is required, you may always reach out to PSW.

In case of any query, please feel free to contact us.

Regards,

Conference Program Committee

on behalf of

Conference Chair

Primary Contact ID: conference@gn.amity.edu

Primary Contact # +91 9654 1393 99

[Amity University Uttar Pradesh | Greater Noida Campus | India](#)

TERMS & CONDITIONS

- All accepted, presented, and duly registered papers of the conference will be submitted to IEEE as "conference material" for inclusion in IEEE Xplore Digital Library, as per IEEE guidelines.
- The conference material will be submitted to IEEE around 30 days after the conduct of the conference, with due process.
- At the time of submission of conference material, if any paper does not fall under the IEEE and conference submission guidelines, will be precluded from the conference material.
- Fee payment for registration is the first and foremost step in the registration process, but fee payment alone does not complete the registration process. In addition to the fee payment in the appropriate category, author(s) must go through the copyright transfer process and must also provide the camera-ready (final version) paper, in consensus with the Conference Program Committee.
- Fee payment for registration does not guarantee that the concerned submission (paper) will be published/ included in IEEE Xplore Digital Library. IEEE has the discretion to include/ accept or exclude/ reject one, more or all submissions (papers) of the conference, if found unfit, after their quality assessment review process or otherwise. In case of rejection of submission(s) at the end of IEEE, any which way, author(s) payer(s) of the submission (paper) will not be entitled to get the refund of registration fee, in any case.
- The corresponding/ first author will solely be responsible for putting the name and details of other authors/ co-authors in the submission. The conference committee explicitly considers that all the authors/ co-authors have contributed to the submission and their consent has been taken to put their name in the submission by the corresponding/ first author.

References

- [1] M. Bari Antor *et al.*, “A Comparative Analysis of Machine Learning Algorithms to Predict Alzheimer’s Disease,” *J. Healthc. Eng.*, vol. 2021, 2021, doi: 10.1155/2021/9917919.
- [2] S. Dogan *et al.*, “Primate brain pattern-based automated Alzheimer’s disease detection model using EEG signals,” *Cogn. Neurodyn.*, vol. 17, no. 3, pp. 647–659, Jun. 2023, doi: 10.1007/s11571-022-09859-2.
- [3] C. H. Chang, C. H. Lin, and H. Y. Lane, “Machine learning and novel biomarkers for the diagnosis of alzheimer’s disease,” *International Journal of Molecular Sciences*, vol. 22, no. 5. MDPI AG, pp. 1–12, Mar. 01, 2021. doi: 10.3390/ijms22052761.
- [4] A. Chakravarthy, B. S. Panda, and S. K. Nayak, “Review and Comparison for Alzheimer’s Disease Detection with Machine Learning Techniques 1,” vol. 27, no. 4, 2023, doi: 10.5123/inj.2023.4.inj42.
- [5] G. Battineni, N. Chintalapudi, F. Amenta, and E. Traini, “A comprehensive machine-learning model applied to magnetic resonance imaging (MRI) to predict Alzheimer’s disease (ad) in older subjects,” *J. Clin. Med.*, vol. 9, no. 7, pp. 1–14, Jul. 2020, doi: 10.3390/jcm9072146.
- [6] J. Bin Bae *et al.*, “Identification of Alzheimer’s disease using a convolutional neural network model based on T1-weighted magnetic resonance imaging,” *Sci. Rep.*, vol. 10, no. 1, Dec. 2020, doi: 10.1038/s41598-020-79243-9.
- [7] X. Tang and J. Liu, “Comparing different algorithms for the course of alzheimer’s disease using machine learning,” *Ann. Palliat. Med.*, vol. 10, no. 9, pp. 9715–9724, Sep. 2021, doi: 10.21037/apm-21-2013.
- [8] R. Kumari, A. Nigam, and S. Pushkar, “Machine learning technique for early detection of Alzheimer’s disease,” *Microsyst. Technol.*, vol. 26, no. 12, pp. 3935–3944, Dec. 2020, doi: 10.1007/s00542-020-04888-5.
- [9] H. Nawaz, M. Maqsood, S. Afzal, F. Aadil, I. Mehmood, and S. Rho, “A deep feature-based real-time system for Alzheimer disease stage detection,” *Multimed. Tools Appl.*, vol. 80, no. 28–29, pp. 35789–35807, Nov. 2021, doi: 10.1007/s11042-020-09087-y.
- [10] S. Koga, A. Ikeda, and D. W. Dickson, “Deep learning-based model for

- diagnosing Alzheimer's disease and tauopathies," *Neuropathol. Appl. Neurobiol.*, vol. 48, no. 1, Feb. 2022, doi: 10.1111/nan.12759.
- [11] N. An, H. Ding, J. Yang, R. Au, and T. F. A. Ang, "Deep ensemble learning for Alzheimer's disease classification," *J. Biomed. Inform.*, vol. 105, May 2020, doi: 10.1016/j.jbi.2020.103411.
 - [12] A. Ebrahimi, S. Luo, and for the A. Disease Neuroimaging Initiative, "Convolutional neural networks for Alzheimer's disease detection on MRI images," *J. Med. Imaging*, vol. 8, no. 02, Apr. 2021, doi: 10.1117/1.jmi.8.2.024503.
 - [13] A. W. Salehi, P. Baglat, and G. Gupta, "Alzheimer's Disease Diagnosis using Deep Learning Techniques," *Int. J. Eng. Adv. Technol.*, vol. 9, no. 3, pp. 874–880, Feb. 2020, doi: 10.35940/ijeat.C5345.029320.
 - [14] B. Yadav Kasula, "International Journal of Sustainable Development in Computing Science A Machine Learning Approach for Differential Diagnosis and Prognostic Prediction in Alzheimer's Disease JOURNAL I N F O." [Online]. Available: www.ijsdcs.com
 - [15] A. Khan and S. Zubair, "An Improved Multi-Modal based Machine Learning Approach for the Prognosis of Alzheimer's disease," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 6, pp. 2688–2706, Jun. 2022, doi: 10.1016/j.jksuci.2020.04.004.
 - [16] S. El-Sappagh, H. Saleh, F. Ali, E. Amer, and T. Abuhmed, "Two-stage deep learning model for Alzheimer's disease detection and prediction of the mild cognitive impairment time," *Neural Comput. Appl.*, vol. 34, no. 17, pp. 14487–14509, Sep. 2022, doi: 10.1007/s00521-022-07263-9.
 - [17] S. Asif Hassan and T. Khan, "A Machine Learning Model to Predict the Onset of Alzheimer Disease using Potential Cerebrospinal Fluid (CSF) Biomarkers," 2017. [Online]. Available: www.ijacsa.thesai.org
 - [18] E. Lella *et al.*, "Machine learning and DWI brain communicability networks for Alzheimer's disease detection," *Appl. Sci.*, vol. 10, no. 3, Feb. 2020, doi: 10.3390/app10030934.
 - [19] J. Venugopalan, L. Tong, H. R. Hassanzadeh, and M. D. Wang, "Multimodal deep learning models for early detection of Alzheimer's disease stage," *Sci. Rep.*, vol. 11, no. 1, Dec. 2021, doi: 10.1038/s41598-

020-74399-w.

- [20] K. N. Rao, B. R. Gandhi, M. V. Rao, S. Javvadi, S. S. Vellela, and S. Khader Basha, "Prediction and Classification of Alzheimer's Disease using Machine Learning Techniques in 3D MR Images," *Int. Conf. Sustain. Comput. Smart Syst. ICSCSS 2023 - Proc.*, no. Icsess, pp. 85–90, 2023, doi: 10.1109/ICSCSS57650.2023.10169550.
- [21] M. R-cnn, "INTELLIGENT SYSTEMS AND APPLICATIONS IN Brain MRI Image Analysis for Alzheimer ' s Disease Diagnosis Using," vol. 12, pp. 137–149, 2024.
- [22] B. Lu *et al.*, "A practical Alzheimer's disease classifier via brain imaging-based deep learning on 85,721 samples," *J. Big Data*, vol. 9, no. 1, Dec. 2022, doi: 10.1186/s40537-022-00650-y.
- [23] A. Yiğit and Z. Işık, "Applying deep learning models to structural MRI for stage prediction of Alzheimer's disease," *Turkish J. Electr. Eng. Comput. Sci.*, vol. 28, no. 1, pp. 196–210, 2020, doi: 10.3906/elk-1904-172.
- [24] S. El-Sappagh, J. M. Alonso, S. M. R. Islam, A. M. Sultan, and K. S. Kwak, "A multilayer multimodal detection and prediction model based on explainable artificial intelligence for Alzheimer's disease," *Sci. Rep.*, vol. 11, no. 1, Dec. 2021, doi: 10.1038/s41598-021-82098-3.
- [25] S. Gao and D. Lima, "A review of the application of deep learning in the detection of Alzheimer's disease," *International Journal of Cognitive Computing in Engineering*, vol. 3. KeAi Communications Co., pp. 1–8, Jun. 01, 2022. doi: 10.1016/j.ijcce.2021.12.002.
- [26] A. A, P. M, M. Hamdi, S. Bourouis, K. Rastislav, and F. Mohamed, "Evaluation of Neuro Images for the Diagnosis of Alzheimer's Disease Using Deep Learning Neural Network," *Front. public Heal.*, vol. 10, p. 834032, 2022, doi: 10.3389/fpubh.2022.834032.

Plagiarism Report

Alzheimer's disease detection using machine learning

ORIGINALITY REPORT

10%

SIMILARITY INDEX

7%

INTERNET SOURCES

9%

PUBLICATIONS

4%

STUDENT PAPERS

PRIMARY SOURCES

1

shropsys.com

Internet Source

1%

2

dspace.univ-ouargla.dz

Internet Source

<1%

3

"Mobile Radio Communications and 5G Networks", Springer Science and Business Media LLC, 2024

Publication

<1%

4

Ketipearachchi C.D., Katugampala K.D.M.N., Kumari P. A. T. H., Sewwandi S. D. I., Sanjeevi Chandrasiri, Lokesha Weerasinghe.
"SoundMind: Doctor Assistive System", 2023 IEEE 17th International Conference on Industrial and Information Systems (ICIIS), 2023

Publication

<1%