AUTOMATED ALZHEIMER DISEASE PREDICTION USING MRI SCANS AND TRANSFER LEARNING TECHNIQUES

Subasty M, Ganesan.T

Anna University/E.G.S.Pillay Engineering College

ABSTRACT

Disorders of the brain are one of the most difficult diseases to cure because of their fragility, the difficulty of performing procedures, and the high costs. On the other hand, the surgery itself does not have to be effective because the results are uncertain. Adults who have hypertension, one of the most common brain illnesses, may have different degrees of memory problems and forgetfulness. Depending on each patient's situation. For these reasons, it's crucial to define memory loss, determine the patient's level of decline, and determine his brain MRI scans are used to identify Alzheimer's disease. In this project, we discuss methods and approaches for diagnosing Alzheimer's disease using deep learning. The suggested approach is utilized to enhance patient care, lower expenses, and enable quick and accurate analysis in sizable investigations. Alzheimer's disease is a progressive neurodegenerative disorder that affects millions of people worldwide. Early detection of the disease can improve patient outcomes, and brain MRI scans have shown promise as a tool for detecting Alzheimer's disease in its early stages. In recent years, deep learning algorithms, such as pre trained model named as VGG16, have been increasingly used in Alzheimer's disease analysis from brain MRI scans. This paper proposes a VGG16-based system for Alzheimer's disease analysis from brain MRI scans. The proposed system involves several steps, including data preprocessing, feature extraction, training the VGG16 model, and evaluating its performance on a test set. The results demonstrate the effectiveness of the proposed pre trained model-based system in accurately detecting Alzheimer's disease from brain MRI scans. The proposed system has the potential to improve early detection and monitoring of Alzheimer's disease, leading to improved patient outcomes.

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVES

The primary objective of this project is to develop a robust and accurate deep learning-based system for the early detection and diagnosis of Alzheimer's disease using brain MRI scans. By leveraging the VGG16 pre-trained convolutional neural network model, the system aims to automate the identification of Alzheimer's disease indicators from MRI images, focusing on structural changes in brain regions such as the hippocampus. The proposed approach emphasizes data preprocessing, augmentation, and normalization to enhance model training and ensure high accuracy, sensitivity, and specificity. Early and precise identification of the disease can lead to timely intervention and improved patient outcomes, reducing the burden on healthcare providers and caregivers. Another key objective is to integrate the proposed system into a clinically viable tool that assists medical professionals in making informed decisions. The model aims to support large-scale screening, enabling quick and consistent assessments without the limitations of manual diagnosis, which can be time-consuming and prone to human error. Additionally, the project seeks to address existing system drawbacks such as high false-positive rates, computational complexity, and limited image support by utilizing advanced deep learning techniques and transfer learning. Ultimately, this system is designed to be scalable, reliable, and capable of evolving with continued research and dataset expansion to enhance the detection and monitoring of Alzheimer's disease.

1.2 SCOPE OF THE PROJECT

The scope of this project involves developing a deep learning-based system for the early detection and classification of Alzheimer’s disease using brain MRI scans. The system utilizes the VGG16 pre-trained neural network model to analyze structural patterns in MRI images, aiming to identify early signs of Alzheimer’s with high accuracy. The project encompasses multiple stages including data collection, image preprocessing (augmentation and normalization), feature extraction, model training, evaluation, and deployment. It also considers the integration of clinical data such as cognitive test scores and medical history to improve the prediction accuracy. This project is intended for implementation in real-world healthcare environments to assist doctors and medical professionals in diagnosing Alzheimer's disease efficiently and reliably. The system is designed to handle large volumes

of MRI data, provide consistent results, and reduce the time and cost associated with manual diagnosis. Furthermore, the project highlights future enhancements like the use of multi-modal imaging (e.g., PET scans), longitudinal patient data, and increased dataset generalizability to improve the model's performance and clinical relevance. Ethical considerations, patient data privacy, and regulatory compliance are also within the project's operational scope.

1.3 METHODOLOGY

The methodology of this project involves using a VGG16-based deep learning approach for the detection of Alzheimer's disease from brain MRI scans. The process begins with the collection and labeling of MRI datasets from both affected and unaffected individuals. These images undergo preprocessing steps such as resizing, normalization, and augmentation to prepare them for model training. The VGG16 model, pre-trained on ImageNet, is fine-tuned using transfer learning to adapt it for Alzheimer's classification tasks. The dataset is divided into training, validation, and testing subsets to evaluate the model's performance. Key features such as cortical thickness and hippocampal volume are extracted to aid classification. Performance metrics like accuracy, precision, recall, F1 score, and AUC are used to assess the model's effectiveness. The final trained model is intended to be deployed in clinical settings, offering fast and accurate diagnostic support to healthcare professionals.

1.4 OVERVIEW OF THE PROJECT

Alzheimer's disease (AD) is the neurodegenerative disease among older adults and affects around 46 million people in the world. The early symptom of the disease is forgetting recent events or conversations. As the disease progresses, it brings severe impairment in memory and loss the ability to take over every day's task. The beginning of the damage takes place in the region of the brain which is responsible in controlling the memory, but the process starts years before the first symptom. The loss of the neuron spreads to other regions and later the brain has shrunk significantly. The disease has the following stages: Mild, Moderate and Severe. The mild stage is the early stage and it includes the problems coming up with the right word or name and losing or misplacing the valuable object. The moderate stage is the middle and longest stage which includes forgetfulness about one's own personal History and confusion about where they are and what day it is. The severe stage is the late stage and includes experience changes in physical abilities and has difficulty in communicating. More than 4 million people are suffering from Alzheimer's and other forms of dementia in India,

which gives the third highest caseload in the world, after China and the United States. India's dementia and Alzheimer's burden is forecast to reach almost 7.5 million at the end of 2030. Predicting the disease priory brings down the risk of death rate. To do so, various methods are used to implement the classification of the disease. The early symptoms of Alzheimer's disease are memory loss, mood changes, poor judgement, social withdrawal, changes in vision. This happens because Alzheimer's disease affects the hippocampus, which plays an important role in memory. One in every 3 seconds a new person somewhere is affected by dementia. At first, it typically destroys neurons and their connections in parts of the brain involved in memory, including the entorhinal cortex and hippocampus. Hence, to understand the changes in the brain need to be studied and explored. It is in this context that MRI images are used as input. and a use a deep learning approach is used to study the variations shown by various factors to calculate the transition from Mild Cognitive Impairment (MCI) to AD. So, when changes are evident in those factors, one can be aware of such changes and take needed medications.

1.5 ORGANIZATION OF THE REPORT

This dissertation presents a deep learning-based framework for the early diagnosis of Alzheimer's disease using brain MRI images. The project emphasizes automating the prediction process through convolutional neural networks (CNN), specifically leveraging the VGG-16 model for classification. The system aims to enhance diagnostic accuracy, reduce human error, and support medical professionals with reliable decision-making tools. The dissertation comprises eight chapters:

- **Chapter 1** introduces the project scope and objectives. It provides a clear overview of the proposed system, emphasizing the importance of early Alzheimer's diagnosis and the role of automated systems in improving healthcare efficiency.
- **Chapter 2** offers a comprehensive literature review. It explores prior research from 2019 to 2024, highlighting the evolution of machine learning and deep learning techniques in medical imaging and neurodegenerative disease detection. This chapter also identifies gaps in traditional diagnostic methods that the proposed system aims to address.
- **Chapter 3** outlines existing systems and their limitations, such as reliance on manual analysis, slow diagnosis, and inconsistent results. It then introduces the proposed

VGG-16-based deep learning model, showcasing its strengths in automated feature learning, consistency, and scalability for real-time analysis.

- **Chapter 4** describes the architectural design of the system in detail. This includes a diagrammatic representation of the workflow from image input to classification output. The chapter also features UML diagrams like use case, activity, class, and sequence diagrams that visually depict the system's structure and logic flow.
- **Chapter 5** details the development environment and technical requirements. It specifies the software platforms, hardware configurations, and tools used in the implementation, along with insights into dataset preparation and preprocessing techniques such as noise filtering and resizing.
- **Chapter 6** focuses on system implementation and module integration. It elaborates on key modules such as image acquisition and preprocessing, feature extraction, CNN classification, and result interpretation. The role of VGG-16 in multi-layered feature analysis is also discussed here.
- **Chapter 7** presents the system evaluation and results. The performance of the model is assessed using metrics such as accuracy, precision, recall, and confusion matrix. Graphs and result snapshots demonstrate the system's effectiveness in detecting early-stage Alzheimer’s.
- **Chapter 8** concludes the dissertation by summarizing the achievements of the project. It reinforces the utility of using deep learning in medical diagnostics and outlines future directions, such as incorporating multi-modal data, expanding the dataset, and enhancing real-time clinical integration.

CHAPTER 2

LITERATURE SURVEY

2.1 TITLE: Volumetric feature-based Alzheimer's disease diagnosis from sMRI data using a convolutional neural network and a deep neural network

AUTHOR: Basher, Abol, et al.

YEAR: 2021

DESCRIPTION: This study introduces a volumetric feature-based approach for diagnosing Alzheimer's Disease (AD) using structural MRI (sMRI) data. The authors designed a hybrid framework combining a Convolutional Neural Network (CNN) and Deep Neural Network (DNN) to improve the accuracy of AD classification. The CNN extracts volumetric features from the sMRI data, which are then processed by the DNN to perform classification. This two-tier structure enables the model to learn high-level representations and intricate spatial dependencies within the brain. The framework was validated on a well-known dataset, and experiments showed it outperformed traditional machine learning techniques and single deep learning models. The research highlighted the significance of integrating volumetric brain information with neural networks to enhance diagnostic accuracy. Additionally, the study discussed preprocessing strategies, including skull stripping and normalization, to ensure robust feature extraction.

TECHNIQUE: Combination of CNN and DNN for volumetric sMRI data analysis

MERIT: Achieves higher accuracy by utilizing both spatial and abstract features

DEMERIT: Requires extensive computational resources and preprocessing steps

2.2 TITLE: Deep residual learning for neuroimaging: an application to predict progression to Alzheimer's disease

AUTHOR: Abrol, Anees, et al.

YEAR: 2020

DESCRIPTION: This research paper explores the use of deep residual networks (ResNet) for predicting the progression from mild cognitive impairment (MCI) to Alzheimer's disease using neuroimaging data. The study addresses the limitations of traditional CNNs by introducing ResNets that allow the training of much deeper neural architectures without the

vanishing gradient problem. The model uses both structural and functional MRI data, creating a rich input space that captures diverse features of brain anatomy and function. The authors argue that deeper networks can extract subtle, disease-relevant features that shallow models often miss. They conduct extensive experiments using cross-validation on a public dataset and report high performance in both sensitivity and specificity. Their model also generalizes well to unseen patient data. In addition to ResNet, the researchers incorporate demographic and clinical features to improve prediction outcomes. The study demonstrates how combining deep learning with neuroimaging biomarkers can pave the way for early diagnosis and personalized treatment strategies for Alzheimer's disease.

TECHNIQUE: Deep residual networks (ResNet) applied to neuroimaging dat

MERIT: Handles deep architectures effectively, improving prediction accuracy

DEMERIT: Training deeper networks requires large labeled datasets

2.3 TITLE: Alzheimer's Disease Diagnosis using Functional and Structural Neuroimaging Modalities

AUTHOR: De Silva, S., S. Dayarathna, and D. Meedeniya

YEAR: 2021

DESCRIPTION: This chapter investigates the application of both structural and functional neuroimaging techniques for diagnosing Alzheimer's Disease. The authors discuss the importance of integrating data from different imaging modalities such as sMRI, PET, and fMRI to provide a comprehensive view of the brain's structure and function. The fusion of multimodal data is shown to enhance diagnostic accuracy, especially in early stages where subtle changes may not be evident in a single modality. The study examines various machine learning and deep learning methods used for preprocessing, feature extraction, dimensionality reduction, and classification. It also delves into challenges related to aligning multi-source data, mitigating noise, and handling varying spatial resolutions. The chapter emphasizes hybrid fusion strategies (data-level, feature-level, and decision-level) and explores their impact on improving sensitivity and specificity. Real-world datasets and experimental results are presented to validate their claims. This comprehensive analysis

serves as a roadmap for building robust, AI-assisted diagnostic systems in clinical environments.

TECHNIQUE: Multimodal neuroimaging with data fusion strategies

MERIT: Improved diagnostic accuracy through comprehensive brain analysis

DEMERIT: Complex preprocessing and data alignment procedures

2.4 TITLE: Diagnosis of Alzheimer's disease and behavioural variant frontotemporal dementia with machine learning-aided neuropsychological assessment using feature engineering and genetic algorithms

AUTHOR: Garcia-Gutierrez, Fernando, et al.

YEAR: 2022

DESCRIPTION: This paper presents a novel approach for distinguishing between Alzheimer's Disease (AD) and behavioral variant Frontotemporal Dementia (bvFTD) using neuropsychological assessment data enhanced by machine learning. The study applies feature engineering techniques to identify critical test results that differentiate between the two diseases. Genetic algorithms are used to optimize feature selection and enhance classification performance. The model is trained on real clinical data and demonstrates strong predictive capability across diverse patient profiles. One of the key strengths of this method lies in its interpretability, as clinicians can understand the decision-making process through the selected features. The study emphasizes the benefits of combining cognitive test scores with computational algorithms for early detection and classification. It also outlines the challenges posed by overlapping symptoms in AD and bvFTD and how machine learning can aid in more accurate diagnoses.

TECHNIQUE: Feature engineering and genetic algorithms for neuropsychological data

MERIT: Efficiently distinguishes between AD and FTD with high interpretability

DEMERIT: Heavily depends on quality and consistency of neuropsychological data

2.5 TITLE: A 3D densely connected convolution neural network with connection-wise attention mechanism for Alzheimer's disease classification

AUTHOR: Zhang, Jie, et al.

YEAR: 2021

DESCRIPTION: This study introduces a 3D densely connected convolutional neural network (DenseNet) enhanced with a connection-wise attention mechanism for classifying Alzheimer's Disease from MRI scans. The 3D architecture allows the model to learn volumetric features that capture detailed spatial patterns across multiple slices of brain scans. The DenseNet structure facilitates feature reuse and strengthens gradient flow, while the attention mechanism selectively enhances important connections. Together, these components enable the model to focus on relevant brain regions typically affected by AD, improving classification performance. The network was trained and validated using public datasets and outperformed standard 2D CNNs and traditional DenseNets. The authors also provide a detailed explanation of the attention layer's role in refining feature maps. The model shows promise for real-world deployment, particularly in clinical settings where high diagnostic accuracy is essential. However, the added complexity of the architecture results in longer training times and increased resource requirements.

TECHNIQUE: 3D DenseNet with connection-wise attention

MERIT: Improves feature relevance and classification accuracy

DEMERIT: Complex architecture increases training time and hardware demands

SUMMARY

The reviewed literature presents a comprehensive overview of recent advancements in Alzheimer's disease (AD) diagnosis using various deep learning and machine learning techniques. Studies have shown the effectiveness of convolutional neural networks (CNNs), deep neural networks (DNNs), residual networks (ResNets), and 3D DenseNet architectures in analyzing structural and functional neuroimaging data, such as MRI and PET scans. Hybrid and multimodal approaches, which integrate both structural and functional data, have demonstrated superior performance in capturing complex brain patterns and improving classification accuracy. Additionally, some works focus on neuropsychological assessments

enhanced by feature engineering and genetic algorithms to distinguish between AD and similar neurodegenerative disorders like frontotemporal dementia. Across the board, these studies highlight the growing importance of data fusion, attention mechanisms, and feature selection in enhancing the diagnostic capability of AI-based systems for early detection and prognosis of Alzheimer's disease.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In the existing system, Alzheimer's disease is often analyzed using features related to other illnesses such as diabetes. Feature selection techniques are applied to identify relevant parameters, which help improve the classification performance of machine learning models. Image processing methods, especially segmentation, are used to separate important areas in brain MRI scans. Techniques like K-means clustering are commonly used to group similar pixel regions in images, making it easier to detect brain abnormalities. These methods rely heavily on manual feature engineering and require domain expertise to choose the right features for accurate diagnosis. Various machine learning algorithms such as K-means clustering, support vector machines (SVM), and random forests have been applied to the Alzheimer's Disease Neuroimaging Initiative (ADNI) dataset. These systems have shown potential in detecting patterns of brain atrophy and predicting the progression of Alzheimer's disease. However, they have some drawbacks including limited support for large image sets, high computational complexity, and a high rate of false positives. These limitations reduce the effectiveness of existing systems, highlighting the need for more accurate and scalable deep learning-based approaches.

3.1.1 DISADVANTAGES OF EXISTING SYSTEM

- Supports only a small set of brain MRI images.
- Uses a fully supervised learning approach.
- Has a high false positive rate in diagnosis.
- Involves complex and time-consuming computations.
- Relies heavily on manual feature selection and engineering.

3.2 PROPOSED SYSTEM

The proposed system aims to improve the early detection and diagnosis of Alzheimer's disease using brain MRI scans by implementing a deep learning approach based on the VGG16 pre-trained neural network model. The system begins with collecting MRI images from individuals with and without Alzheimer's, followed by preprocessing steps like image resizing, normalization, and augmentation to prepare the data. Feature extraction focuses on key brain regions such as the hippocampus and cortical structures, which are known to show early signs of the disease. Transfer learning is used to fine-tune the VGG16 model on the Alzheimer's dataset, enabling it to detect disease-specific patterns even with limited data. Once trained, the model is evaluated using metrics such as accuracy, precision, recall, F1 score, and AUC to ensure high diagnostic performance. The final system can be deployed in clinical settings to assist healthcare professionals by providing fast and accurate predictions. The approach also considers future improvements like integrating multi-modal imaging (e.g., PET scans), combining MRI with clinical data, and conducting longitudinal studies to monitor disease progression. By reducing manual effort, increasing scalability, and improving diagnostic accuracy, the proposed system represents a more efficient and reliable solution.

3.2.1 ADVANTAGES OF PROPOSED SYSTEM

- Provides fast and accurate detection of Alzheimer's disease.
- Reduces manual effort through automated deep learning techniques.
- Uses a balanced dataset to improve model training efficiency.
- Achieves better performance in terms of sensitivity, specificity, and accuracy.
- Supports early diagnosis, leading to improved patient treatment and outcomes.

CHAPTER 4

DEVELOPMENT REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

- Processor : Intel processor 2.6.0 GHZ
- RAM : 2 GB
- Hard disk : 160 GB

- Compact Disk : 650 Mb
- Keyboard : Standard keyboard
- Monitor : 15 inch color monitor

4.2 SOFTWARE REQUIREMENTS

- Operating system : Windows OS
- Front End : Python
- IDLE : Python 3.7 IDLE
- Libraries : Tensor flow, KERAS

4.3 SOFTWARE DESCRIPTION

Frond End: Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation. Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach. While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is not considered a compliment in the Python culture."Python's philosophy rejects the Perl "there is more than one way to do it" approach

to language design in favour of "there should be one—and preferably only one—obvious way to do it".

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of CPython that would offer marginal increases in speed at the cost of clarity.[When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. CPython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter. An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name a tribute to the British comedy group Monty Python and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard for and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called un pythonic. Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoneers. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source

level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python's initial development was spearheaded by Guido van Rossum in the late 1980s. Today, it is developed by the Python Software Foundation. Because Python is a multiparadigm language, Python programmers can accomplish their tasks using different styles of programming: object oriented, imperative, functional or reflective. Python can be used in Web development, numeric programming, game development, serial port access and more.

There are two attributes that make development time in Python faster than in other programming languages:

1. Python is an interpreted language, which precludes the need to compile code before executing a program because Python does the compilation in the background. Because Python is a high-level programming language, it abstracts many sophisticated details from the programming code. Python focuses so much on this abstraction that its code can be understood by most novice programmers.
2. Python code tends to be shorter than comparable codes. Although Python offers fast development times, it lags slightly in terms of execution time. Compared to fully compiling languages like C and C++, Python programs execute slower. Of course, with the processing speeds of computers these days, the speed differences are usually only observed in benchmarking tests, not in real-world operations. In most cases, Python is already included in Linux distributions and Mac OS X machines.

TENSORFLOW LIBARIES IN PYTHON

TensorFlow is an open-source machine learning framework developed by Google Brain Team. It is one of the most popular libraries for building and training machine learning models, especially deep neural networks. TensorFlow allows developers to build complex models with ease, including image and speech recognition, natural language processing, and more. One of the key features of TensorFlow is its ability to handle large-scale datasets and complex computations, making it suitable for training deep neural networks. It allows for parallelization of computations across multiple CPUs or GPUs, allowing for faster training times. TensorFlow also provides a high-level API called Keras that simplifies the process of building and training models. TensorFlow offers a wide range of tools and libraries that make it easy to integrate with other Python libraries and frameworks. It has built-in support for data preprocessing and visualization, making it easy to prepare data for training and analyze model performance. One of the major advantages of TensorFlow is its ability to deploy models to a variety of platforms, including mobile devices and the web.

Graph-based computation: TensorFlow uses a graph-based computation model, which allows for efficient execution of computations across multiple devices and CPUs/GPUs.

Automatic differentiation: TensorFlow provides automatic differentiation, which allows for efficient computation of gradients for use in backpropagation algorithms.

High-level APIs: TensorFlow provides high-level APIs, such as Keras, that allow developers to quickly build and train complex models with minimal code.

Preprocessing and data augmentation: TensorFlow provides a range of tools for preprocessing and data augmentation, including image and text preprocessing, data normalization, and more.

Distributed training: TensorFlow supports distributed training across multiple devices, CPUs, and GPUs, allowing for faster training times and more efficient use of resources.

Model deployment: TensorFlow allows for easy deployment of models to a variety of platforms, including mobile devices and the web.

Visualization tools: TensorFlow provides a range of visualization tools for analyzing model performance, including TensorBoard, which allows for real-time visualization of model training and performance.

Back End: My SQL

MySQL is the world's most used open source relational database management system (RDBMS) as of 2008 that run as a server providing multi-user access to a number of databases. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation.

MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP open source web application software stack—LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python." Free-software-open source projects that require a full-featured database management system often use MySQL.For commercial use, several paid editions are available, and offer additional functionality. Applications which use MySQL databases include: TYPO3, Joomla, Word Press, phpBB, MyBB, Drupal and other software built on the LAMP software stack. MySQL is also used in many high-profile, large-scale World Wide Web products, including Wikipedia, Google(though not for searches), ImagebookTwitter, Flickr, Nokia.com, and YouTube.

Inter images

MySQL is primarily an RDBMS and ships with no GUI tools to administer MySQL databases or manage data contained within the databases. Users may use the included command line tools, or use MySQL "front-ends", desktop software and web applications that create and manage MySQL databases, build database structures, back up data, inspect status, and work with data records. The official set of MySQL front-end tools, MySQL Workbench is actively developed by Oracle, and is freely available for use.

Graphical

The official MySQL Workbench is a free integrated environment developed by MySQL AB, that enables users to graphically administer MySQL databases and visually design database structures. MySQL Workbench replaces the previous package of software, MySQL GUI Tools. Similar to other third-party packages, but still considered the authoritative MySQL frontend, MySQL Workbench lets users manage database design &

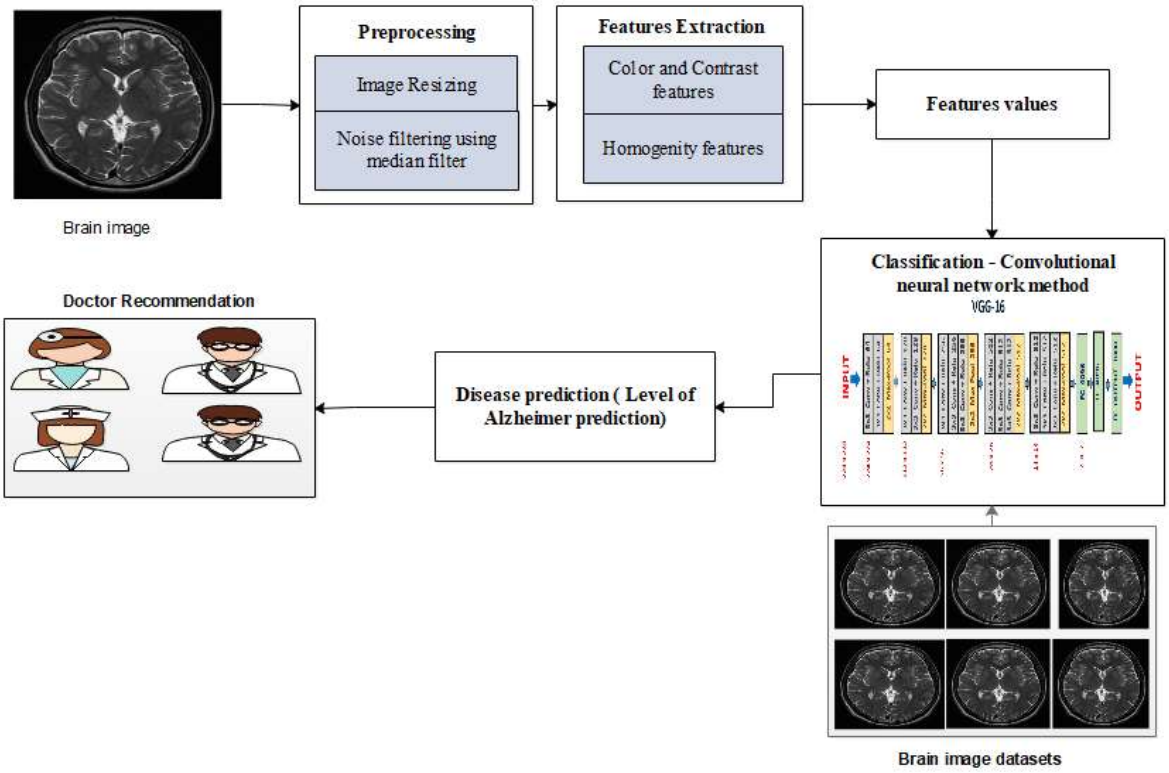
modeling, SQL development (replacing MySQL Query Browser) and Database administration (replacing MySQL Administrator).MySQL Workbench is available in two editions, the regular free and open source Community Edition which may be downloaded from the MySQL website, and the proprietary Standard Edition which extends and improves the feature set of the Community Edition.

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE DESIGN

In this architecture, we can train and test the brain MRI image. In training phase, we can collect the brain images from KAGGLE source. And construct the VGG16 CNN algorithm with VGG16 model for future testing phase. And in testing phase, we can input the brain image and perform image resizing and noise filtering techniques to eliminate the noises in images. Finally classify the image with trained model and then predict the Alzheimer with severity level and also precaution details



5.1.1 DESCRIPTION

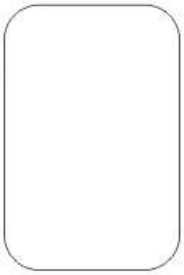
The architecture for early prediction of Alzheimer's disease using brain MRI images begins with the input stage, where raw MRI scans are fed into the system. These images first undergo a thorough preprocessing phase to prepare them for analysis. The preprocessing

includes resizing all images to a consistent dimension, which ensures uniformity across the dataset and allows the deep learning model to process the data efficiently. Additionally, noise filtering is applied using a median filter, a technique that effectively reduces unwanted distortions and artifacts within the images. This step is crucial because it enhances the overall quality of the MRI scans, making it easier for the subsequent stages to extract meaningful and accurate features related to brain structure. Following preprocessing, the system focuses on feature extraction, which is a critical step in identifying the subtle changes in brain patterns caused by Alzheimer's disease. During this stage, important image attributes such as color intensity, contrast differences, and homogeneity are carefully analyzed and extracted. These features highlight variations in brain tissue and structural abnormalities that may indicate the presence or progression of Alzheimer's. By capturing these key characteristics, the system ensures that the deep learning model receives rich and relevant information, enabling it to differentiate between normal and affected brain images with greater precision. The extracted features are then fed into a deep convolutional neural network, specifically the VGG-16 model, which is renowned for its ability to learn hierarchical and complex patterns from image data. Through a series of convolutional layers, the VGG-16 network processes the features to identify intricate spatial patterns and abnormalities associated with different stages of Alzheimer's disease. The model performs classification by assigning the input brain MRI images to appropriate categories, such as mild, moderate, or severe Alzheimer's condition. The final output not only provides a disease prediction but also supports healthcare providers by generating recommendations for further diagnosis and treatment. This automated architecture enhances diagnostic accuracy, reduces reliance on manual interpretation, and aids medical professionals in making timely and informed decisions for patient care.

5.2 DATA FLOW DIAGRAM

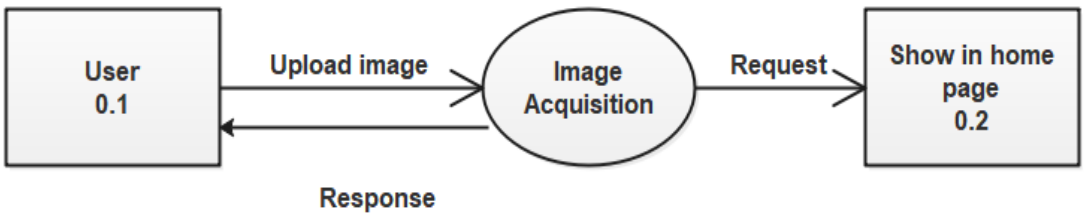
A two-dimensional diagram explains how data is processed and transferred in a system. The graphical depiction identifies each source of data and how it interacts with other data sources to reach a common output. Individuals seeking to draft a data flow diagram must identify external inputs and outputs, determine how the inputs and outputs relate to each other, and explain with graphics how these connections relate and what they result in. This type of diagram helps business development and design teams visualize how data is processed and identify or improve certain aspects.

Data flow Symbols:

Symbol	Description
	An **entity**. A source of data or a destination for data.
	A **process** or task that is performed by the system.
	A **data store**, a place where data is held between processes.
	A **data flow**.

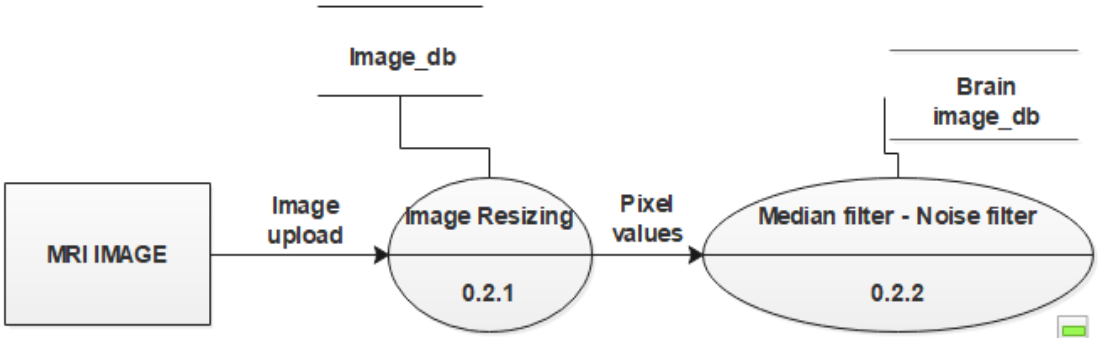
LEVEL 0

The Level 0 DFD shows how the system is divided into 'sub-systems' (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.



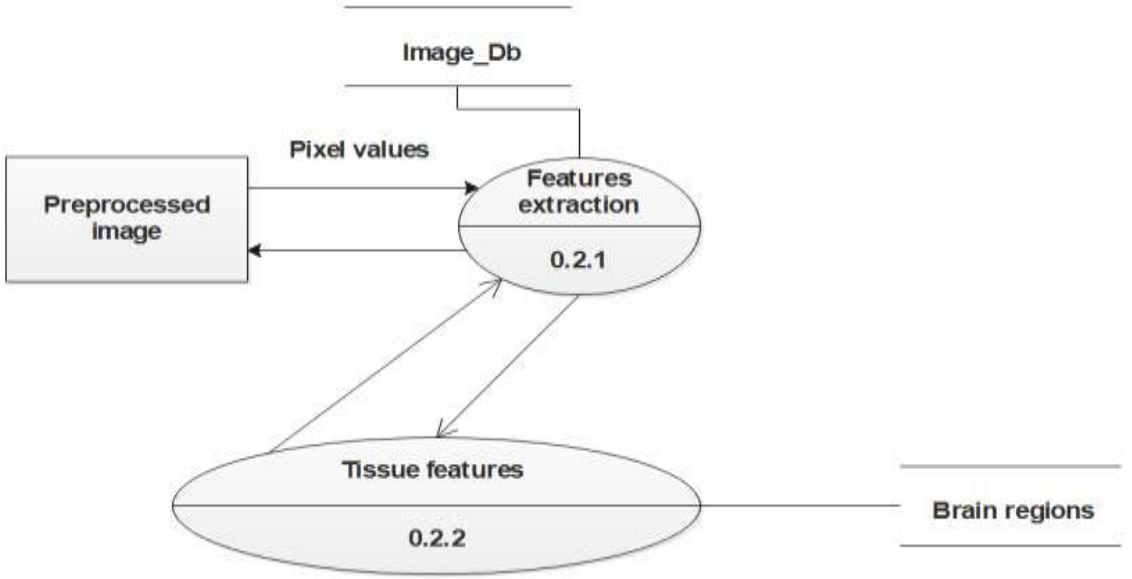
LEVEL 1

The next stage is to create the Level 1 Data Flow Diagram. This highlights the main functions carried out by the system. As a rule, to describe the system was using between two and seven functions - two being a simple system and seven being a complicated system. This enables us to keep the model manageable on screen or paper.



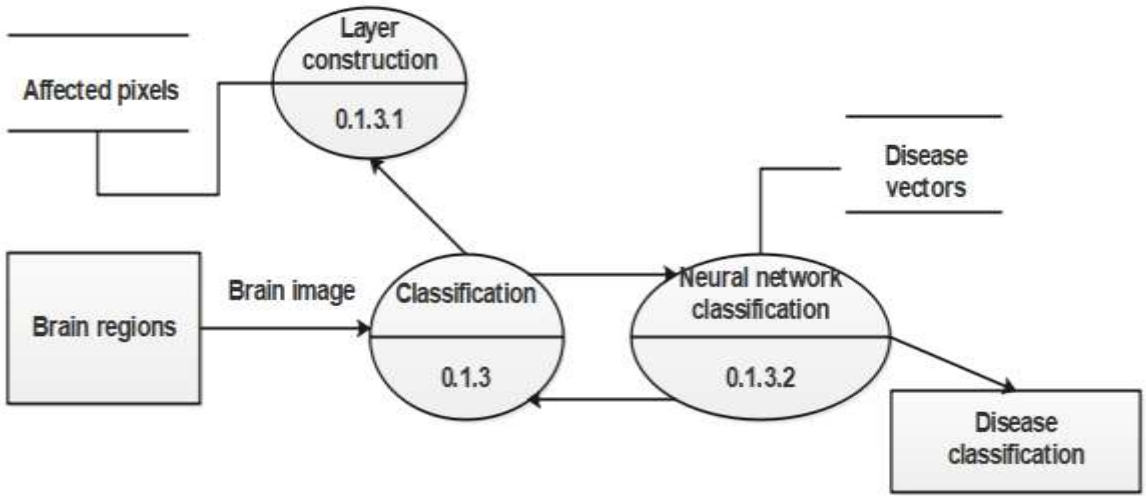
LEVEL 2

A Data Flow Diagram (DFD) tracks processes and their data paths within the business or system boundary under investigation. A DFD defines each domain boundary and illustrates the logical movement and transformation of data within the defined boundary. The diagram shows 'what' input data enters the domain, 'what' logical processes the domain applies to that data, and 'what' output data leaves the domain. Essentially, a DFD is a tool for process modeling and one of the oldest.



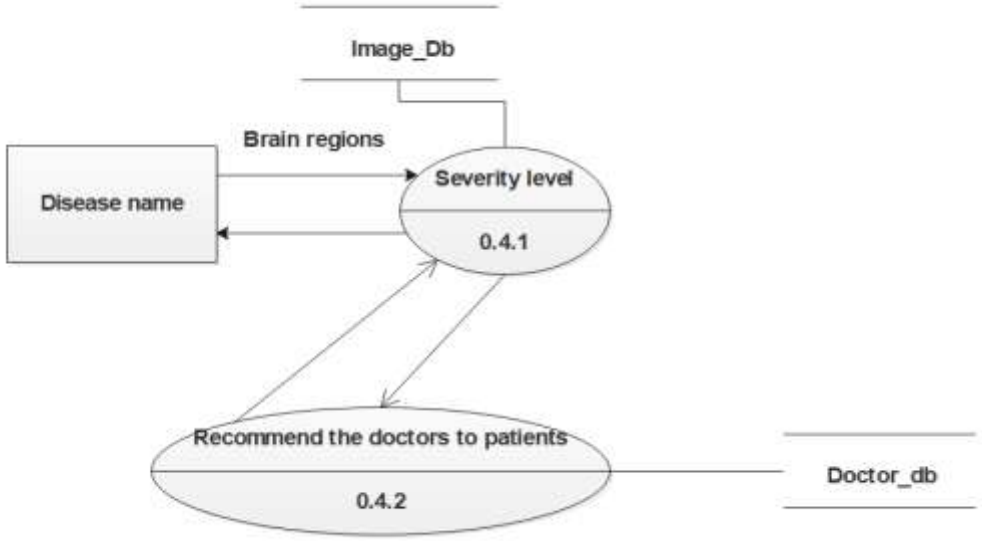
LEVEL-3

A data flow diagram (DFD) is a graphical representation of the flow of data through an information system. A DFD shows the flow of data from data sources and data stores to processes, and from processes to data stores and data sinks. DFDs are used for modeling and analyzing the flow of data in data processing systems, and are usually accompanied by a data dictionary, an entity-relationship model, and a number of process descriptions.



LEVEL-4

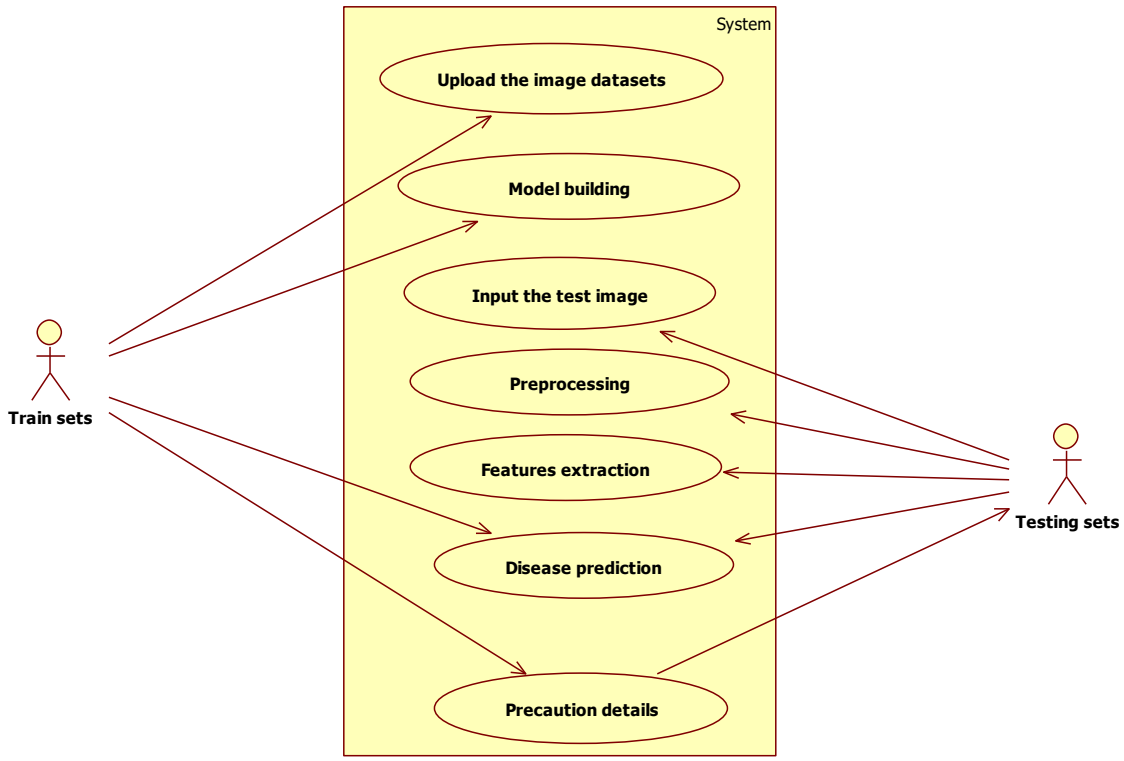
At this level, the focus shifts towards the internal components and structures that support the processes defined in Level 3. Individual sub-processes from Level 3 might be further broken down to reveal specific modules, units, or procedures. Data flows at this stage often represent the movement of data between these internal components and any local data stores they utilize. This level starts to bridge the gap between the logical data flow and the potential physical implementation of the system.



5.3 UML DIAGRAMS

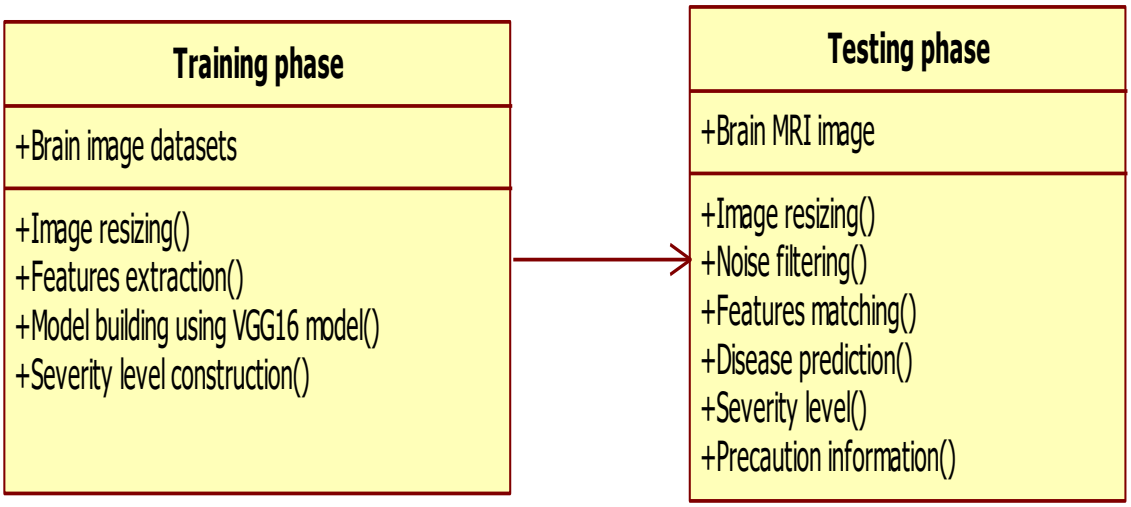
5.3.1 USE CASE DIAGRAM

In its most basic form, a use case diagram is a depiction of a user's interaction with the system that illustrates the connection between the user and the many use cases that the user is involved in. A "system" in this sense refers to something that is being created or run, like a website. The "actors" are individuals or groups functioning inside the system in designated roles.



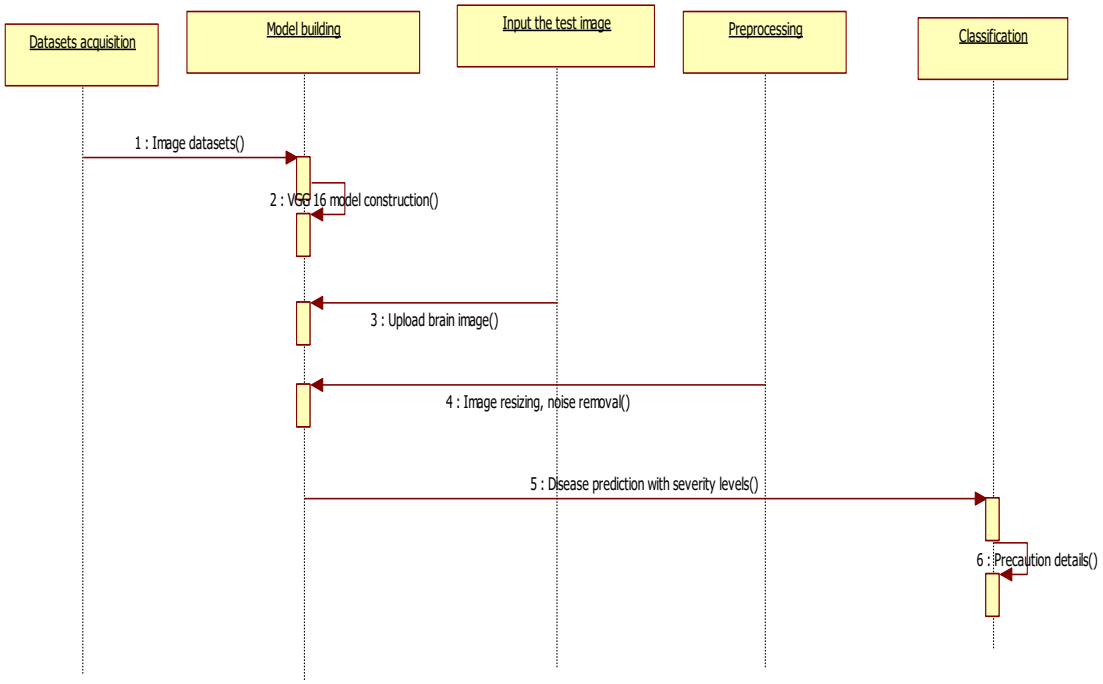
5.3.2 CLASS DIAGRAM

A class diagram, as defined by the Unified Modeling Language (UML), is a kind of static structural diagram that illustrates a system's classes, properties, functions, and interactions between objects. The fundamental component of object-oriented modeling is the class diagram. It is utilized for both technical modeling which converts the models into computer code and general conceptual modeling of the applications systematic.



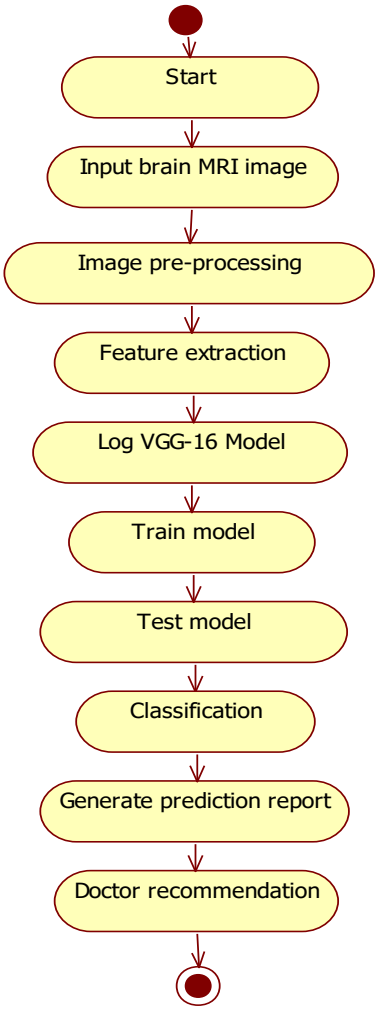
5.3.3 SEQUENCE DIAGRAM

An object's interactions are arranged chronologically in a sequence diagram. It shows the classes and objects that are a part of the scenario as well as the messages that are passed between the objects in order for the scenario to work. Sequence diagrams are commonly linked to the realizations of use cases in the Logical View of the system that is being developed. Event diagrams or event scenarios are other names for sequence diagrams.



5.3.4 ACTIVITY DIAGRAM

The activity diagram shows a unique kind of state diagram in which the majority of states are action states and the majority of transitions are brought about by the fulfillment of actions in the source states. One may refer to the action as a system operation. As a result, the control flow is transferred across operations. This flow may occur concurrently, forked, or sequentially. Activity diagrams use a variety of features to address various forms of flow control.



CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 MODULES LIST

- BRAIN IMAGE AUGUMENTATION
- IMAGE NORMALIZATION
- FEATURES EXTRACTION
- PRETRAINED NEURAL NETWORK
- ALZHEMIER LEVEL IDENTIFICATION

6.2 MODULES DESCRIPTION

6.2.1 BRAIN IMAGE ACQUISITION:

The status of the brain can be examined using an MRI to help predict abnormalities and cerebral activities. In this work, an Automatic Diagnostic Tool (ADT) was created with the intention of analyzing and classifying normal and Alzheimer class MRI signal patterns. We can enter MRI image data in this module. This Kaggle dataset is a collection of MRI scans taken from young patients with untreatable Alzheimer's. After discontinuing anti-Alzheimer mediation, subjects were observed for up to several days in order to describe their data and determine whether they were candidates for surgical surgery.

6.2.2 IMAGE NORMALIZATION

Data preprocessing, which is an integral step in the data mining process, is the transformation or erasure of data before use in order to ensure or improve performance. Rubbish in, and garbage out is especially true for projects using data and machine learning. Data collection techniques are usually not well controlled, which results in out-of-range figures, impossible data combinations, missing information, etc. Inaccurate positive results from data analysis might result from not adequately checking for these problems. As a result, each analysis must be preceded by a review of the data's presentation or quality. The most critical part of a machine learning project often becomes data preparation, especially in computational biology. Serial MRI images are initially split in the before stage by a rolling time window without crossing over. The MRI data is then subjected to a wavelet transformation to produce a collection of signals.

6.2.3 FEATURES EXTRACTION

From preprocessed data, we can extract the temporal or frequency domain features in this module. It contains "mean," "variety," "kurtosis," "skewness," as well as other features for classification in the future. Mean: It can be described as the average of a N sample EEG signal.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} Y_i$$

Standard deviation: A standard deviation is the variation in data from a signal's mean value.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(Y_i - \mu)^2}$$

Kurtosis: It's a statistical phenomenon that imparts a peaked quality to time series data. Kurtosis is derived from

$$k = \frac{1}{N}\sum_{i=1}^{N}(\frac{y_{i-\mu}}{\sigma})^4$$

Skewness: It gauges the symmetry of a signal's distribution and is generated as

$$skewness = \frac{1}{N}\sum_{i=1}^{N}(\frac{y_{i-\mu}}{\sigma})^3$$

6.2.4 PRETRAINED NEURAL NETWORK

In the process of diagnosing Alzheimer's disease utilizing MRI recordings, a variety of comment operations must be carried out on the outputs of a training of VGG16 in order to obtain the offer enhanced again for test MRI scan image. A VGG16 consists of input, a layer, and multiple hidden layers. In a VGG16's hidden layers, convolution, pooling, and fully connected layers are frequently observed. Following a convolutional layer's operation on the input, the subsequent layer receives the outcome. Convolution mimics the response of a single neuron to visual stimuli. In convolutional networks, the output of a neural cluster at one level can be combined with a nerve cell at a higher level using either local or global pooling layers. In mean pooling, the mean value from each neural cell in the previous layers is utilized. Through fully connected layers, every neuron in one layer can communicate with every neuron in every other layer. The classic multi-layer feed-forward neural network and the VGG16 layers are conceptually equivalent. For the study of high-dimensional data, CNNs are unquestionably better than traditional classifiers. To control and reduce the overall number of parameters, convolutional layers of VGG16 employ a parameter-sharing strategy. The network's variable and computation count, in addition to the representation's various studies have demonstrated, are gradually reduced through the use of a pooling layer, providing the network more fitting control.VGG 16 is a model for a 16-layer CNN model. It is still considered one of today’s best and most effective models. Instead of having numerous parameters, the VGG 16 model architecture focuses on ConvNet layers with a 3 × 3 kernel

size. The significance of this model lies in the fact that its values are freely available online and may be downloaded for use in one's systems and applications. When compared to other developed comprehensives, it is noted for its simplicity. This model's minimum expected input image size is 224 × 224 pixels with three channels. In neural networks, optimization algorithms are used to evaluate whether a neuron must be engaged or not, by determining the weighted sum of input. The need for kernel function arises from inducing non-linearity into the output neuron. A neural network's neurons function together with weight, bias, and the related training procedure. The neurons' link weights are adjusted based on the output inaccuracy. The input layer and the activation function add non-linearity to artificial neural input, allowing it to learn and accomplish complex tasks. So, our proposed work overcomes outlier's separation in MRI data classification with features extraction. Based on classification we can predict the level of Alzheimer disease

6.2.5 ALZHEIMER LEVEL IDENTIFICATION

This module interprets the classification results from the VGG-16 model to determine the stage or severity of Alzheimer's disease in the patient. It categorizes MRI images into distinct classes, such as normal, mild cognitive impairment, moderate Alzheimer's, or severe Alzheimer's, based on learned patterns. The identification helps in assessing the progression of the disease, enabling timely medical intervention. Outputs from this module can be used to generate comprehensive reports that aid doctors in treatment planning and monitoring. Accurate level identification is vital for personalized patient care and for tracking the effectiveness of therapies over time. This module integrates the model's predictions with clinical decision-making processes. It also supports follow-up recommendations and continuous patient management.

CHAPTER 7

SYSTEM TESTING

7.1 TESTING

Testing is a set activity that can be planned and conducted systematically. Testing begins at the module level and work towards the integration of entire computers based system. Nothing is complete without testing, as it is vital success of the system.

Testing Objectives:

There are several rules that can serve as testing objectives, they are

1. Testing is a process of executing a program with the intent of finding an error
2. A good test case is one that has high probability of finding an undiscovered error.
3. A successful test is one that uncovers an undiscovered error.

If testing is conducted successfully according to the objectives as stated above, it would uncover errors in the software. Also testing demonstrates that software functions appear to the working according to the specification, that performance requirements appear to have been met.

There are three ways to test a program

1. For Correctness
2. For Implementation efficiency
3. For Computational Complexity.

Tests used for implementation efficiency attempt to find ways to make a correct program faster or use less storage. It is a code-refining process, which reexamines the implementation phase of algorithm development. Tests for computational complexity amount to an experimental analysis of the complexity of an algorithm or an experimental comparison of two or more algorithms, which solve the same problem.

The data is entered in all forms separately and whenever an error occurred, it is corrected immediately. A quality team deputed by the management verified all the necessary documents and tested the Software while entering the data at all levels.

7.2 TYPES OF TESTING

The development process involves various types of testing. Each test type addresses a specific testing requirement. The most common types of testing involved in the development process are:

- Unit Test
- Functional Test
- Integration Test
- White box Test
- Black box Test
- System Test
- Validation Test
- Acceptance Test

Unit Testing:

The first test in the development process is the unit test. The source code is normally divided into modules, which in turn are divided into smaller units called units. These units have specific behavior. The test done on these units of code is called unit test. Unit test depends upon the language on which the project is developed. Unit tests ensure that each unique path of the project performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Functional Testing:

Functional test can be defined as testing two or more modules together with the intent of finding defects, demonstrating that defects are not present, verifying that the module performs its intended functions as stated in the specification and establishing confidence that a program does what it is supposed to do.

Integration Testing:

In integration testing modules are combined and tested as a group. Modules are typically code modules, individual applications, source and destination applications on a network, etc.

Integration Testing follows unit testing and precedes system testing. Testing after the product is code complete. Betas are often widely distributed or even distributed to the public at large in hopes that they will buy the final product when it is released.

White Box Testing:

Testing based on an analysis of internal workings and structure of a piece of software. This testing can be done sing the percentage value of load and energy. The tester should know what exactly is done in the internal program. It includes techniques such as Branch Testing and Path Testing. White box testing also called as Structural Testing or Glass Box Testing.

Black Box Testing:

In block box testing without knowledge of the internal workings of the item being tested. Tests are usually functional. This testing can be done by the user who has no knowledge of how the shortest path is found.

System Testing

System testing is defined as testing of a complete and fully integrated software product. This testing falls in black-box testing wherein knowledge of the inner design of the code is not a pre-requisite and is done by the testing team. It is the final test to verify that the product to be delivered meets the specifications mentioned in the requirement document. It should investigate both functional and non-functional requirements.

Validation Testing

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements. Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

Acceptance Testing

This is a type of testing done by users, customers, or other authorised entities to determine application/software needs and business processes. Acceptance testing is the most important phase of testing as this decides whether the client approves the application/software or not. It may involve functionality, usability, performance, and U.I of the application. It is also

known as user acceptance testing (UAT), operational acceptance testing (OAT), and end-user testing.

7.3 SOFTWARE TESTING STRATEGIES

In order to make sure that the programme satisfies its intended criteria and performs as expected, software testing strategies are detailed blueprints that direct the testing procedure throughout a software development project. Various testing approaches are used, depending on the particulars of the project:

Waterfall Testing Strategy: The requirements, design, implementation, testing, and deployment phases of the waterfall model are sequentially approached, with each stage being finished before going on to the next. With a focus on validation against predetermined requirements, this strategy's testing is thorough and well-documented.

V-Model Testing Strategy: The V-Model, which is a development and testing phase relationship model, is an extension of the Waterfall model. To guarantee that requirements are verified early in the process, there is a testing phase that corresponds with each development phase.

Iterative Testing Strategy: This approach, which incorporates incremental testing with gradual additions of modules or components, is perfect for iterative development. Early flaw detection and adaptable modifications in response to feedback are made possible by it.

Exploratory Testing Strategy: Without preset test cases, testers use their experience and instincts to investigate the product, which is a helpful method for identifying bugs that scripted tests, could miss.

Automated Testing Strategy: This approach is especially helpful for regression testing, repetitive jobs, and large-scale projects since it emphasises the use of automated testing tools and scripts to carry out test cases.

The nature of the project, the development process, the available funds, and the amount of time all play a role in selecting a testing approach.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

This project demonstrates the effective use of deep learning, specifically the VGG-16 convolutional neural network, to analyze brain MRI images for the early detection and classification of Alzheimer's disease. By leveraging advanced preprocessing techniques such as image augmentation and normalization, along with robust feature extraction, the proposed system achieves improved accuracy and reliability in identifying disease presence and severity. The application of transfer learning through the pretrained VGG-16 model significantly reduces training time while enhancing performance, making it a promising tool for medical image analysis in the field of neurodegenerative diseases. Early and accurate diagnosis of Alzheimer's disease is crucial for timely intervention and patient care management. The developed system not only automates and speeds up the diagnostic process but also assists healthcare professionals by providing consistent and objective analysis of brain scans. While further validation on larger and more diverse datasets is necessary, this approach holds significant potential to improve clinical outcomes, reduce healthcare costs, and ultimately contribute to better quality of life for individuals affected by Alzheimer's disease.

8.2 FUTURE WORK

- **Integration of Multi-Modal Data:** Future work can incorporate additional data sources such as PET scans, genetic information, and clinical test results alongside MRI images. Combining multiple modalities can improve diagnostic accuracy and provide a more comprehensive understanding of the disease progression.
- **Improved Model Architectures:** Experimenting with newer and more advanced deep learning architectures like EfficientNet, ResNet, or transformer-based models may enhance feature extraction and classification performance.
- **Explainable AI Techniques:** Incorporating explainability methods such as Grad-CAM or SHAP can help clinicians understand the model's decision-making process, increasing trust and acceptance in medical settings.

- **Longitudinal Analysis:** Developing models that analyze sequential MRI scans over time could provide better insights into disease progression and help predict future cognitive decline more accurately.
- **Larger and More Diverse Datasets:** Expanding the dataset to include more patients from different demographics and stages of Alzheimer's will improve model generalizability and reduce bias.
- **Real-Time Clinical Integration:** Building user-friendly interfaces and integrating the system with hospital information systems can facilitate seamless adoption in clinical workflows.
- **Mobile and Edge Deployment:** Optimizing the model for deployment on mobile devices or edge hardware can enable point-of-care diagnostics in remote or resource-limited settings.
- **Continuous Learning Systems:** Implementing systems that update and improve automatically with new incoming data can keep the model relevant and accurate over time.

APPENDIX

A1. SOURCE CODE

```
# importing libraries


import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import random

import glob  # to find files
```

```
# Seaborn library for bar chart

import seaborn as sns


# Libraries for TensorFlow

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.preprocessing import image

from tensorflow.keras import models, layers


# Library for Transfer Learning

from tensorflow.keras.applications import VGG16

from keras.applications.vgg16 import preprocess_input


print("Importing libraries completed.")


path = 'Datas/'


train_folder = path + "train/"

train_normal_dir = train_folder + "MildDemented/"

train_pneu_dir = train_folder + "NonDemented/"

train_pneu_dir1 = train_folder + "VeryMildDemented/"


# test directory

test_folder = path + "test/"

test_normal_dir = test_folder + "MildDemented/"

test_pneu_dir = test_folder + "NonDemented/"
```

```
test_pneu_dir1 = test_folder + "VeryMildDemented/"


# validation directory

val_folder = path + "Val/"


val_normal_dir = val_folder + "MildDemented/"

val_pneu_dir = val_folder + "NonDemented/"

val_pneu_dir1 = val_folder + "VeryMildDemented/"


# variables for image size

img_width = 100

img_height = 100


# variable for model

batch_size = 64

epochs = 10


print("Variable declaration completed.")


# listing the folders containing images


# Train Dataset

train_class_names = os.listdir(train_folder)

print("Train class names: %s" % (train_class_names))

# print("\n")
```

```
# Test Dataset
test_class_names = os.listdir(test_folder)
print("Test class names: %s" % (test_class_names))
# print("\n")

# Validation Dataset
val_class_names = os.listdir(val_folder)
print("Validation class names: %s" % (val_class_names))

print("\nDataset class name listing completed.")



# declaration of functions

# Function get name of xray type
def Get_Xray_Type(argument):
    switcher = {
        "MildDemented": "MildDemented",
        "NonDemented": "NonDemented",
        "VeryMildDemented": "VeryMildDemented",
    }
    return switcher.get(argument, "Invalid image")
```

```
print("Declaration of functions completed.")


# Analysis of Train, Test and Validation directory


# find all files, our files has extension jpeg

train_normal_cases = glob.glob(train_normal_dir + '*jpg')

train_pneu_cases = glob.glob(train_pneu_dir + '*jpg')

train_pneu_cases1 = glob.glob(train_pneu_dir1 + '*jpg')


test_normal_cases = glob.glob(test_normal_dir + '*jpg')

test_pneu_cases = glob.glob(test_pneu_dir + '*jpg')

test_pneu_cases1 = glob.glob(test_pneu_dir1 + '*jpg')


val_normal_cases = glob.glob(val_normal_dir + '*jpg')

val_pneu_cases = glob.glob(val_pneu_dir + '*jpg')

val_pneu_cases1 = glob.glob(val_pneu_dir1 + '*jpg')

# create lists for train, test & validation cases, create labels as well

train_list = []

test_list = []

val_list = []


for x in train_normal_cases:

    train_list.append([x, "MildDemented"])
```

```
for x in train_pneu_cases:
    train_list.append([x, "NonDemented"])

for x in train_pneu_cases1:
    train_list.append([x, "VeryMildDemented"])

for x in test_normal_cases:
    test_list.append([x, "MildDemented"])

for x in test_pneu_cases:
    test_list.append([x, "NonDemented"])

for x in test_pneu_cases1:
    test_list.append([x, "VeryMildDemented"])

for x in val_normal_cases:
    val_list.append([x, "MildDemented"])

for x in val_pneu_cases:
    val_list.append([x, "NonDemented"])

for x in val_pneu_cases1:
    val_list.append([x, "VeryMildDemented"])
```

```
# create dataframes

train_df = pd.DataFrame(train_list, columns=['image', 'Diagnos'])

print(train_df.shape)

test_df = pd.DataFrame(test_list, columns=['image', 'Diagnos'])

print(test_df.shape)

val_df = pd.DataFrame(val_list, columns=['image', 'Diagnos'])

print(val_df.shape)


plt.figure(figsize=(20, 5))


plt.subplot(1, 3, 1)

# sns.countplot(train_df['Diagnos'])

# sns.distplot(train_df, y='Diagnos')


ax = sns.countplot(y=train_df['Diagnos'], data=train_df)

plt.title('Train data')


plt.subplot(1, 3, 2)

ax = sns.countplot(y=test_df['Diagnos'], data=train_df)

plt.title('Test data')


plt.subplot(1, 3, 3)


ax = sns.countplot(y=val_df['Diagnos'], data=train_df)

plt.title('Validation data')
```

```
plt.show()


plt.figure(figsize=(20, 8))
for i, img_path in enumerate(train_df[train_df['Diagnos'] == "MildDemented"][0:4]['image']):
    plt.subplot(2, 4, i + 1)
    plt.axis('off')
    img = plt.imread(img_path)
    plt.imshow(img, cmap='gray')
    plt.title('MildDemented')


for i, img_path in enumerate(train_df[train_df['Diagnos'] == "NonDemented"][0:4]['image']):
    plt.subplot(2, 4, 4 + i + 1)
    plt.axis('off')
    img = plt.imread(img_path)
    plt.imshow(img, cmap='gray')
    plt.title('Normal')
plt.show()
# Declaring variables
x = []  # to store array value of the images
y = []  # to store the labels of the images


for folder in os.listdir(train_folder):
    image_list = os.listdir(train_folder + "/" + folder)
```

```
    for img_name in image_list:

        # Loading images

        img = image.load_img(train_folder + "/" + folder + "/" + img_name, target_size=(img_width, img_height))


        # Converting to arrary

        img = image.img_to_array(img)


        # Transfer Learning: this is to apply preprocess of VGG16 model to our images before passing it to VGG16

        img = preprocess_input(img)  # Optional step


        # Appending the arrarys

        x.append(img)  # appending image array

        y.append(train_class_names.index(folder))  # appending class index to the array


print("Preparing Training Dataset Completed.")


# Preparing validation images data (image array and class name) for processing


# Declaring variables

val_images = []

val_images_Original = []

val_image_label = []  # to store the labels of the images


for folder in os.listdir(val_folder):
```

```
    image_list = os.listdir(val_folder + "/" + folder)

    for img_name in image_list:

        # Loading images

        img = image.load_img(val_folder + "/" + folder + "/" + img_name, target_size=(img_width, img_height))


        # Converting to arrarys

        img = image.img_to_array(img)


        # Saving original images, will be used just for display at the end

        val_images_Original.append(img.copy())


        # Transfer Learning: this is to apply preprocess of VGG16 to our images before passing it to VGG16

        img = preprocess_input(img)  # Optional step


        # Appending arrays

        val_images.append(img)  # appending image array

        val_image_label.append(val_class_names.index(folder))


print("Preparing Validation Dataset Completed.")


# Preparing validation images data (image array and class name) for processing


# Declaring variables

test_images = []
```

```
test_images_Original = []

test_image_label = []  # to store the labels of the images


for folder in os.listdir(test_folder):
    image_list = os.listdir(test_folder + "/" + folder)
    for img_name in image_list:
        # Loading images
        img = image.load_img(test_folder + "/" + folder + "/" + img_name, target_size=(img_width, img_height))


        # Converting to arrarys
        img = image.img_to_array(img)


        # Saving original images, will be used just for display at the end
        test_images_Original.append(img.copy())


        # Transfer Learning: this is to apply preprocess of VGG16 to our images before passing it to VGG16
        img = preprocess_input(img)  # Optional step


        # Appending arrays
        test_images.append(img)  # appending image array
        test_image_label.append(test_class_names.index(folder))


print("Preparing Test Dataset Completed.")
```

```
# Verifying the output


# Training Dataset

print("Training Dataset")


x = np.array(x)  # Converting to np arrary to pass to the model

print(x.shape)


y = to_categorical(y)  # onehot encoding of the labels

# print(y)

print(y.shape)


# ===========


# Test Dataset

print("Test Dataset")


test_images = np.array(test_images)

print(test_images.shape)


test_image_label = to_categorical(test_image_label)  # onehot encoding of the labels)

print(test_image_label.shape)


# ===========
```

```
# Validation Dataset

print("Validation Dataset")


val_images = np.array(val_images)

print(val_images.shape)


val_image_label = to_categorical(val_image_label)  # onehot encoding of the labels)

print(val_image_label.shape)


print("Summary of default VGG16 model.\n")


# we are using VGG16 for transfer learnin here. So we have imported it

from tensorflow.keras.applications import VGG16


model_vgg16 = VGG16(weights='imagenet')

model_vgg16.summary()


print("Summary of Custom VGG16 model.\n")

input_layer = layers.Input(shape=(img_width, img_height, 3))

model_vgg16 = VGG16(weights='imagenet', input_tensor=input_layer, include_top=False)

model_vgg16.summary()

last_layer = model_vgg16.output

flatten = layers.Flatten()(last_layer)

output_layer = layers.Dense(3, activation='softmax')(flatten)

model = models.Model(inputs=input_layer, outputs=output_layer)
```

```
model.summary()
for layer in model.layers[:-1]:
    layer.trainable = False
model.summary()

from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2, random_state=5)
print("Splitting data for train and test completed.")

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

print("Model compilation completed.")

history2 = model.fit(xtrain, ytrain, epochs=epochs, batch_size=batch_size, verbose=True, validation_data=(xtest, ytest))

print("Fitting the model completed.")

model.save("Vggmodel")

acc = history2.history['accuracy']
val_acc = history2.history['val_accuracy']
epochs = range(len(acc))
```

```
plt.plot(epochs, acc, label='Training Accuracy')
plt.plot(epochs, val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Plot Model Loss
loss_train = history2.history['loss']
loss_val = history2.history['val_loss']
plt.plot(epochs, loss_train, label='Training Loss')
plt.plot(epochs, loss_val, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
from flask import Flask, render_template, flash, request, session, send_file
from flask import render_template, redirect, url_for, request
from werkzeug.utils import secure_filename
import mysql.connector
```

```
import sys


import pickle


app = Flask(__name__)

app.config['DEBUG']

app.config['SECRET_KEY'] = '7d441f27d441f27567d441f2b6176a'




@app.route("/")

def homepage():

    return render_template('index.html')




@app.route("/Home")

def Home():

    return render_template('index.html')




@app.route("/DoctorLogin")

def DoctorLogin():

    return render_template('DoctorLogin.html')




@app.route("/NewDoctor")
```

```
def NewDoctor():
    return render_template('NewDoctor.html')


@app.route("/AdminLogin")
def AdminLogin():
    return render_template('AdminLogin.html')


@app.route("/UserLogin")
def UserLogin():
    return render_template('UserLogin.html')


@app.route("/NewUser")
def NewUser():
    return render_template('NewUser.html')


@app.route("/Cancer")
def Cancer():
    return render_template('Cancer.html')


@app.route("/Diabetes")
```

```
def Diabetes():
    return render_template('Diabetes.html')



@app.route("/Heart")
def Heart():
    return render_template('Heart.html')



@app.route("/adminlogin", methods=['GET', 'POST'])
def adminlogin():
    error = None
    if request.method == 'POST':
        if request.form['uname'] == 'admin' and request.form['password'] == 'admin':

            conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
            # cursor = conn.cursor()
            cur = conn.cursor()
            cur.execute("SELECT * FROM regtb ")
            data = cur.fetchall()
            return render_template('AdminHome.html', data=data)

        else:
```

```
        alert = 'Username or Password is wrong'

        return render_template('goback.html', data=alert)



@app.route("/AdminHome")

def AdminHome():

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

    # cursor = conn.cursor()

    cur = conn.cursor()

    cur.execute("SELECT * FROM regtb ")

    data = cur.fetchall()


    return render_template('AdminHome.html', data=data)


@app.route("/AdminUserInfo")

def AdminUserInfo():

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

    # cursor = conn.cursor()

    cur = conn.cursor()

    cur.execute("SELECT * FROM doctortb ")

    data = cur.fetchall()


    return render_template('AdminUserInfo.html', data=data)
```

```
@app.route("/AdminAssignInfo")
def AdminAssignInfo():
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
    # cursor = conn.cursor()
    cur = conn.cursor()
    cur.execute("SELECT * FROM drugtb ")
    data = cur.fetchall()

    return render_template('AdminAssignInfo.html', data=data)


@app.route("/DoctorUserInfo")
def DoctorUserInfo():
    dname = session['dname']

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
    # cursor = conn.cursor()
    cur = conn.cursor()
    cur.execute("SELECT * FROM apptb where DoctorName='" + dname + "' ")
    data = cur.fetchall()

    return render_template('DoctorUserInfo.html', data=data)
```

```
@app.route("/DoctorAssignInfo")
def DoctorAssignInfo():
    dname = session['dname']

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
    # cursor = conn.cursor()
    cur = conn.cursor()
    cur.execute("SELECT * FROM drugtb where DoctorName='" + dname + "' ")
    data = cur.fetchall()

    return render_template('DoctorAssignInfo.html', data=data)


@app.route("/doclogin", methods=['GET', 'POST'])
def doclogin():
    if request.method == 'POST':
        username = request.form['uname']
        password = request.form['password']
        session['dname'] = request.form['uname']

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
        cursor = conn.cursor()
```

```
    cursor.execute("SELECT * from doctortb where username='" + username + "' and Password='" + password + "'")

    data = cursor.fetchone()

    if data is None:


        data1 = 'Username or Password is wrong'

        return render_template('goback.html', data=data1)




    else:

        print(data[0])

        session['uid'] = data[0]

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

        # cursor = conn.cursor()

        cur = conn.cursor()

        cur.execute("SELECT * FROM doctortb where username='" + username + "' and Password='" + password + "'")

        data = cur.fetchall()


        return render_template('DoctorHome.html', data=data)




@app.route("/searchid")

def searchid():

    user = request.args.get('user')
```

```
    session['user'] = user

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

    cur = conn.cursor()

    cur.execute(

        "SELECT  *  FROM apptb where  username='" + str(user) + "'")

    data = cur.fetchall()

    print(data)


    return render_template('AdminAssign.html', data=data)



@app.route("/assigndrug", methods=['GET', 'POST'])

def assigndrug():

    if request.method == 'POST':

        uname = request.form['UserName']

        phone = request.form['Phone']

        email = request.form['Email']

        dname = session['dname']

        medi = request.form['Medicine']

        other = request.form['Other']

        file = request.files['file']

        file.save("static/upload/" + file.filename)

        Adate = request.form['Adate']
```

```
        conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

        cursor = conn.cursor()

        cursor.execute(

            "INSERT INTO  drugtb VALUES ('" + uname + "','" + phone + "','" + email + "','" + dname + "','" + medi + "','" + other + "','" + file.filename + "','" + Adate + "')")

        conn.commit()

        conn.close()


        # return 'file register successfully'

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

        # cursor = conn.cursor()

        cur = conn.cursor()

        cur.execute("SELECT * FROM drugtb where DoctorName='" + dname + "' ")

        data = cur.fetchall()


    return render_template('DoctorAssignInfo.html', data=data)



@app.route("/newuser", methods=['GET', 'POST'])

def newuser():

    if request.method == 'POST':

        name1 = request.form['name']

        gender1 = request.form['gender']

        Age = request.form['age']
```

```
        email = request.form['email']

        pnumber = request.form['phone']

        address = request.form['address']


        uname = request.form['uname']

        password = request.form['psw']

        loc = request.form['loc']


        conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

        cursor = conn.cursor()

        cursor.execute(

            "INSERT INTO regtb VALUES ('" + name1 + "','" + gender1 + "','" + Age + "','" + email + "','" + pnumber + "','" + address + "','" + uname + "','" + password + "','" + loc + "')")

        conn.commit()

        conn.close()

        # return 'file register successfully'


    return render_template('UserLogin.html')



@app.route("/newdoctor", methods=['GET', 'POST'])

def newcoor():

    if request.method == 'POST':

        name1 = request.form['name']

        gender1 = request.form['gender']
```

```
        Age = request.form['age']

        email = request.form['email']

        pnumber = request.form['phone']

        address = request.form['address']

        special = request.form['special']

        loc = request.form['loc']


        uname = request.form['uname']

        password = request.form['psw']


        conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

        cursor = conn.cursor()

        cursor.execute(

            "INSERT INTO doctortb VALUES ('" + name1 + "','" + gender1 + "','" + Age + "','" + email + "','" + pnumber + "','" + address + "','" + special + "','" + uname + "','" + password + "','" + loc + "')")

        conn.commit()

        conn.close()


    data1 = 'Record Saved'

    return render_template('goback.html', data=data1)



@app.route("/userlogin", methods=['GET', 'POST'])

def userlogin():
```

```
    error = None
    if request.method == 'POST':
        username = request.form['uname']
        password = request.form['password']
        session['uname'] = request.form['uname']

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
        cursor = conn.cursor()
        cursor.execute("SELECT * from regtb where username='" + username + "' and Password='" + password + "'")
        data = cursor.fetchone()
        if data is None:

            data1 = 'Username or Password is Incorrect!'
            return render_template('goback.html', data=data1)



        else:
            print(data[0])
            session['uid'] = data[0]
            session['loca'] = data[8]

            conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
```

```
        # cursor = conn.cursor()

        cur = conn.cursor()

        cur.execute("SELECT * FROM regtb where username='" + username + "' and Password='" + password + "'")

        data = cur.fetchall()


        return render_template('UserHome.html', data=data)



@app.route("/ViewDoctor")

def ViewDoctor():

    return render_template('UserAppointment.html')



@app.route("/predict", methods=['GET', 'POST'])

def predict():

    if request.method == 'POST':

        file = request.files['file']

        file.save('static/upload/Test.jpg')

        fname = 'static/upload/Test.jpg'


        import warnings

        warnings.filterwarnings('ignore')


        import tensorflow as tf
```

```
import numpy as np

import os

from keras.preprocessing import image

base_dir = 'Datas/train/'

catgo = os.listdir(base_dir)

print(catgo)


classifierLoad = tf.keras.models.load_model('Vggmodel/model.h5')

test_image = image.load_img('static/upload/Test.jpg', target_size=(100, 100))

test_image = np.expand_dims(test_image, axis=0)

result = classifierLoad.predict(test_image)

print(result)

ind = np.argmax(result)


# print(catgo[ind])


if result[0][0] == 1:

    print("VeryMildDemented")

    out = "VeryMildDemented"


elif result[0][1] == 1:

    print("MildDemented")

    out = "MildDemented"


elif result[0][2] == 1:
```

```
            print("NonDemented")

            out = "NonDemented"


        if out == "NonDemented":

            print('Normal')

            return render_template('UserAppointment.html', pre=out)

        else:

            conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

            cur = conn.cursor()

            cur.execute("SELECT * FROM doctortb ")

            data = cur.fetchall()

            session['out'] = out

            return render_template('UserAppointment.html', pre=out, data=data)



@app.route("/UserSearch", methods=['GET', 'POST'])

def UserSearch():

    if request.method == 'POST':

        special = request.form['special']

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

        # cursor = conn.cursor()

        cur = conn.cursor()

        cur.execute("SELECT * FROM doctortb where Specialist='" + special + "'")

        data = cur.fetchall()
```

```
    return render_template('UserAppointment.html', data=data)


@app.route("/UserAppointment")
def UserAppointment():
    uname = session['uname']

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
    # cursor = conn.cursor()
    cur = conn.cursor()
    cur.execute("SELECT * FROM  apptb where UserName='" + uname + "' ")
    data = cur.fetchall()

    return render_template('UserAppointmentinfo.html', data=data)


@app.route("/UserAssignDrugInfo")
def UserAssignDrugInfo():
    uname = session['uname']
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
    cur = conn.cursor()
    cur.execute("SELECT * FROM  drugtb where UserName='" + uname + "'  ")
    data = cur.fetchall()
```

```
    return render_template('UserAssignDrugInfo.html', data=data)



@app.route("/Appointment")
def Appointment():
    dusername = request.args.get('id')
    import datetime
    date = datetime.datetime.now().strftime('%Y-%m-%d')
    uname = session['uname']
    dise = session['out']

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM doctortb where UserNAme='" + dusername + "'")
    data = cursor.fetchone()

    if data:
        spec = data[6]




    else:
```

```
        return 'Incorrect username / password !'


    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
    cursor = conn.cursor()
    cursor.execute("SELECT  *  FROM regtb where  UserNAme='" + uname + "'")
    data = cursor.fetchone()


    if data:
        mobile = data[4]
        email = data[3]



    else:


        return 'Incorrect username / password !'


    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')
    cursor = conn.cursor()
    cursor.execute(
        "INSERT INTO  apptb VALUES ('" + uname + "','" + mobile + "','" + email + "','" + dusername + "','" + date + "','" + spec + "','"+ dise  +"')")
    conn.commit()
    conn.close()
```

```
    data1 = 'Record Saved'

    return render_template('goback.html', data=data1)



@app.route('/download')

def download():

    id = request.args.get('id')


    conn = mysql.connector.connect(user='root', password='', host='localhost', database='2doctorapdbPy')

    cursor = conn.cursor()

    cursor.execute("SELECT  *  FROM drugtb where  id = '" + str(id) + "'")

    data = cursor.fetchone()

    if data:

        filename = "static\\upload\\" + data[7]


        return send_file(filename, as_attachment=True)


    else:

        return 'Incorrect username / password !'



@app.route("/search", methods=['GET', 'POST'])

def search():

    if request.method == 'POST':
```

```
        date = request.form['date']


        conn = mysql.connector.connect(user='root', password='', host='localhost',
database='2doctorapdbPy')

        # cursor = conn.cursor()

        cur = conn.cursor()

        cur.execute("SELECT * FROM assigntb where Lastdate='" + date + "'")

        data = cur.fetchall()


        return render_template('Notification.html', data=data)



if __name__ == '__main__':

    app.run(debug=True, use_reloader=True)
```

A2. SCREEN SHOTS

Importing libraries completed.

Variable declaration completed.

Train class names: ['MildDemented', 'NonDemented', 'VeryMildDemented']

Test class names: ['MildDemented', 'NonDemented', 'VeryMildDemented']

Validation class names: ['MildDemented', 'NonDemented', 'VeryMildDemented']

Dataset class name listing completed.

Declaration of functions completed.

(2007, 2)

(1045, 2)

(757, 2)

Model: "vgg16"

_____

Layer (type)	Output Shape	Param #

=

input_1 (InputLayer) [(None, 224, 224, 3)] 0

block1_conv1 (Conv2D) (None, 224, 224, 64) 1792

block1_conv2 (Conv2D) (None, 224, 224, 64) 36928

block1_pool (MaxPooling2D) (None, 112, 112, 64) 0

block2_conv1 (Conv2D) (None, 112, 112, 128) 73856

block2_conv2 (Conv2D) (None, 112, 112, 128) 147584

block2_pool (MaxPooling2D) (None, 56, 56, 128) 0

block3_conv1 (Conv2D) (None, 56, 56, 256) 295168

block3_conv2 (Conv2D) (None, 56, 56, 256) 590080

block3_conv3 (Conv2D) (None, 56, 56, 256) 590080

block3_pool (MaxPooling2D) (None, 28, 28, 256) 0

block4_conv1 (Conv2D) (None, 28, 28, 512) 1180160

block4_conv2 (Conv2D) (None, 28, 28, 512) 2359808

block4_conv3 (Conv2D) (None, 28, 28, 512) 2359808

block4_pool (MaxPooling2D) (None, 14, 14, 512) 0

block5_conv1 (Conv2D) (None, 14, 14, 512) 2359808

block5_conv2 (Conv2D) (None, 14, 14, 512) 2359808

block5_conv3 (Conv2D) (None, 14, 14, 512) 2359808

block5_pool (MaxPooling2D) (None, 7, 7, 512) 0

flatten (Flatten) (None, 25088) 0

fc1 (Dense) (None, 4096) 102764544

fc2 (Dense) (None, 4096) 16781312

predictions (Dense) (None, 1000) 4097000

==========

=

Total params: 138,357,544

Trainable params: 138,357,544

Non-trainable params: 0

Summary of Custom VGG16 model.

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 100, 100, 3)]	0
block1_conv1 (Conv2D)	(None, 100, 100, 64)	1792
block1_conv2 (Conv2D)	(None, 100, 100, 64)	36928
block1_pool (MaxPooling2D)	(None, 50, 50, 64)	0
block2_conv1 (Conv2D)	(None, 50, 50, 128)	73856
block2_conv2 (Conv2D)	(None, 50, 50, 128)	147584
block2_pool (MaxPooling2D)	(None, 25, 25, 128)	0
block3_conv1 (Conv2D)	(None, 25, 25, 256)	295168
block3_conv2 (Conv2D)	(None, 25, 25, 256)	590080
block3_conv3 (Conv2D)	(None, 25, 25, 256)	590080
block3_pool (MaxPooling2D)	(None, 12, 12, 256)	0

block4_conv1 (Conv2D) (None, 12, 12, 512) 1180160

block4_conv2 (Conv2D) (None, 12, 12, 512) 2359808

block4_conv3 (Conv2D) (None, 12, 12, 512) 2359808

block4_pool (MaxPooling2D) (None, 6, 6, 512) 0

block5_conv1 (Conv2D) (None, 6, 6, 512) 2359808

block5_conv2 (Conv2D) (None, 6, 6, 512) 2359808

block5_conv3 (Conv2D) (None, 6, 6, 512) 2359808

block5_pool (MaxPooling2D) (None, 3, 3, 512) 0

===

=

Total params: 14,714,688

Trainable params: 14,714,688

Non-trainable params: 0

_____

Model: "model"

_____

Layer (type) Output Shape Param #

==========

=

input_2 (InputLayer) [(None, 100, 100, 3)] 0

block1_conv1 (Conv2D) (None, 100, 100, 64) 1792

block1_conv2 (Conv2D) (None, 100, 100, 64) 36928

block1_pool (MaxPooling2D) (None, 50, 50, 64) 0

block2_conv1 (Conv2D) (None, 50, 50, 128) 73856

block2_conv2 (Conv2D) (None, 50, 50, 128) 147584

block2_pool (MaxPooling2D) (None, 25, 25, 128) 0

block3_conv1 (Conv2D) (None, 25, 25, 256) 295168

block3_conv2 (Conv2D) (None, 25, 25, 256) 590080

block3_conv3 (Conv2D) (None, 25, 25, 256) 590080

block3_pool (MaxPooling2D) (None, 12, 12, 256) 0

block4_conv1 (Conv2D) (None, 12, 12, 512) 1180160

block4_conv2 (Conv2D) (None, 12, 12, 512) 2359808

block4_conv3 (Conv2D) (None, 12, 12, 512) 2359808

block4_pool (MaxPooling2D) (None, 6, 6, 512) 0

block5_conv1 (Conv2D) (None, 6, 6, 512) 2359808

block5_conv2 (Conv2D) (None, 6, 6, 512) 2359808

block5_conv3 (Conv2D) (None, 6, 6, 512) 2359808

block5_pool (MaxPooling2D) (None, 3, 3, 512) 0

flatten (Flatten) (None, 4608) 0

dense (Dense) (None, 3) 13827

===

=

Total params: 14,728,515

Trainable params: 14,728,515

Non-trainable params: 0

_____

Model: "model"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 100, 100, 3)]	0
block1_conv1 (Conv2D)	(None, 100, 100, 64)	1792
block1_conv2 (Conv2D)	(None, 100, 100, 64)	36928
block1_pool (MaxPooling2D)	(None, 50, 50, 64)	0
block2_conv1 (Conv2D)	(None, 50, 50, 128)	73856
block2_conv2 (Conv2D)	(None, 50, 50, 128)	147584
block2_pool (MaxPooling2D)	(None, 25, 25, 128)	0
block3_conv1 (Conv2D)	(None, 25, 25, 256)	295168
block3_conv2 (Conv2D)	(None, 25, 25, 256)	590080
block3_conv3 (Conv2D)	(None, 25, 25, 256)	590080

block3_pool (MaxPooling2D) (None, 12, 12, 256) 0

block4_conv1 (Conv2D) (None, 12, 12, 512) 1180160

block4_conv2 (Conv2D) (None, 12, 12, 512) 2359808

block4_conv3 (Conv2D) (None, 12, 12, 512) 2359808

block4_pool (MaxPooling2D) (None, 6, 6, 512) 0

block5_conv1 (Conv2D) (None, 6, 6, 512) 2359808

block5_conv2 (Conv2D) (None, 6, 6, 512) 2359808

block5_conv3 (Conv2D) (None, 6, 6, 512) 2359808

block5_pool (MaxPooling2D) (None, 3, 3, 512) 0

flatten (Flatten) (None, 4608) 0

dense (Dense) (None, 3) 13827

===

=

Total params: 14,728,515

Trainable params: 13,827

Non-trainable params: 14,714,688

Splitting data for train and test completed.

Model compilation completed.

Epoch 1/10

26/26 [==========] - 49s 2s/step - loss: 4.2504 - accuracy: 0.4748 - val_loss: 2.6244 - val_accuracy: 0.5896

Epoch 2/10

26/26 [==========] - 42s 2s/step - loss: 1.7076 - accuracy: 0.6131 - val_loss: 1.4822 - val_accuracy: 0.6119

Epoch 3/10

26/26 [==========] - 43s 2s/step - loss: 1.1159 - accuracy: 0.6779 - val_loss: 1.2842 - val_accuracy: 0.6443

Epoch 4/10

26/26 [==========] - 43s 2s/step - loss: 0.9723 - accuracy: 0.6984 - val_loss: 1.1751 - val_accuracy: 0.6940

Epoch 5/10

26/26 [==========] - 42s 2s/step - loss: 0.6792 - accuracy: 0.7495 - val_loss: 0.9435 - val_accuracy: 0.7264

Epoch 6/10

26/26 [==========] - 42s 2s/step - loss: 0.3842 - accuracy: 0.8517 - val_loss: 0.6975 - val_accuracy: 0.7761

Fitting the model completed.

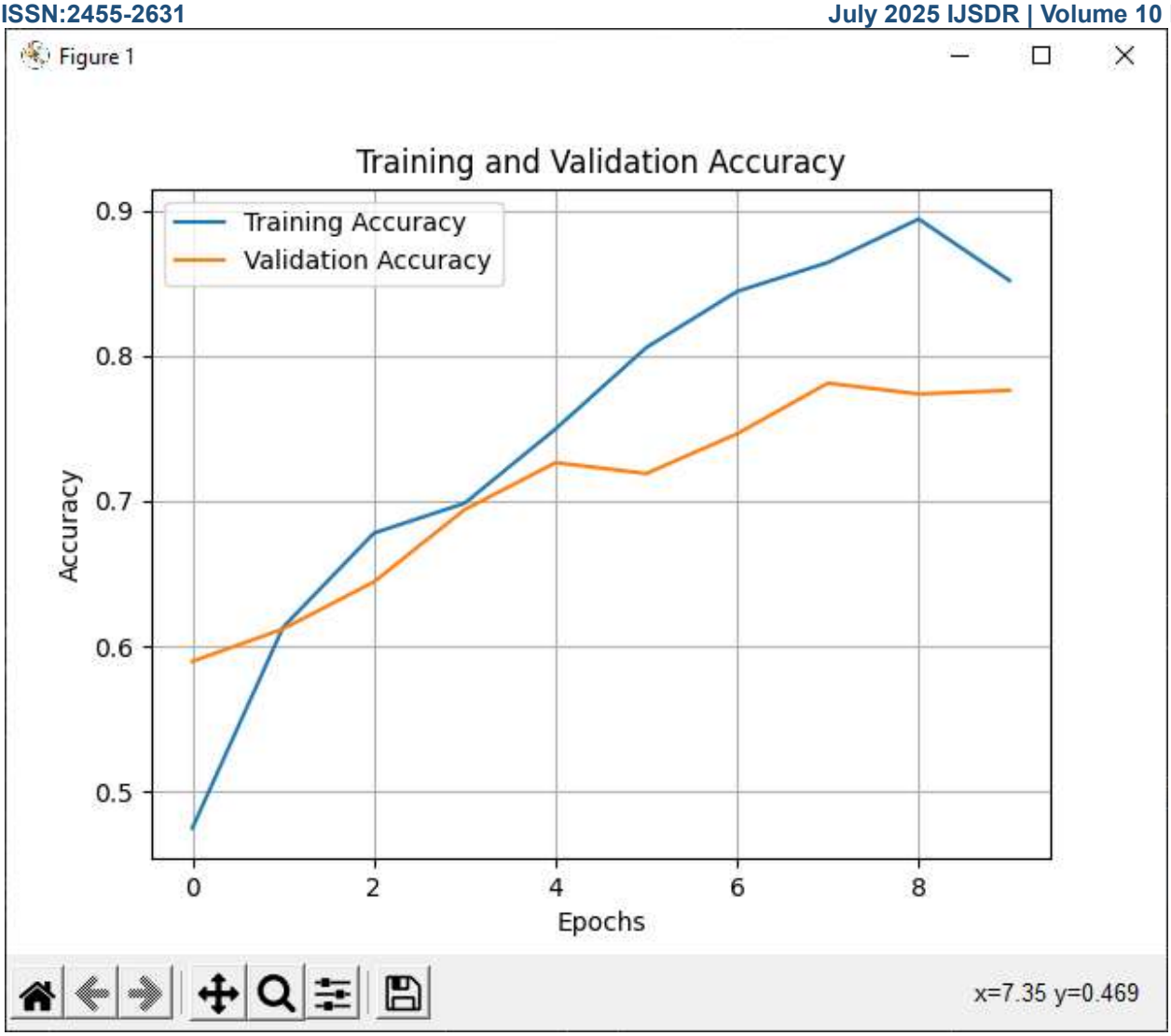


Figure 1
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8
x=7.35 y=0.469

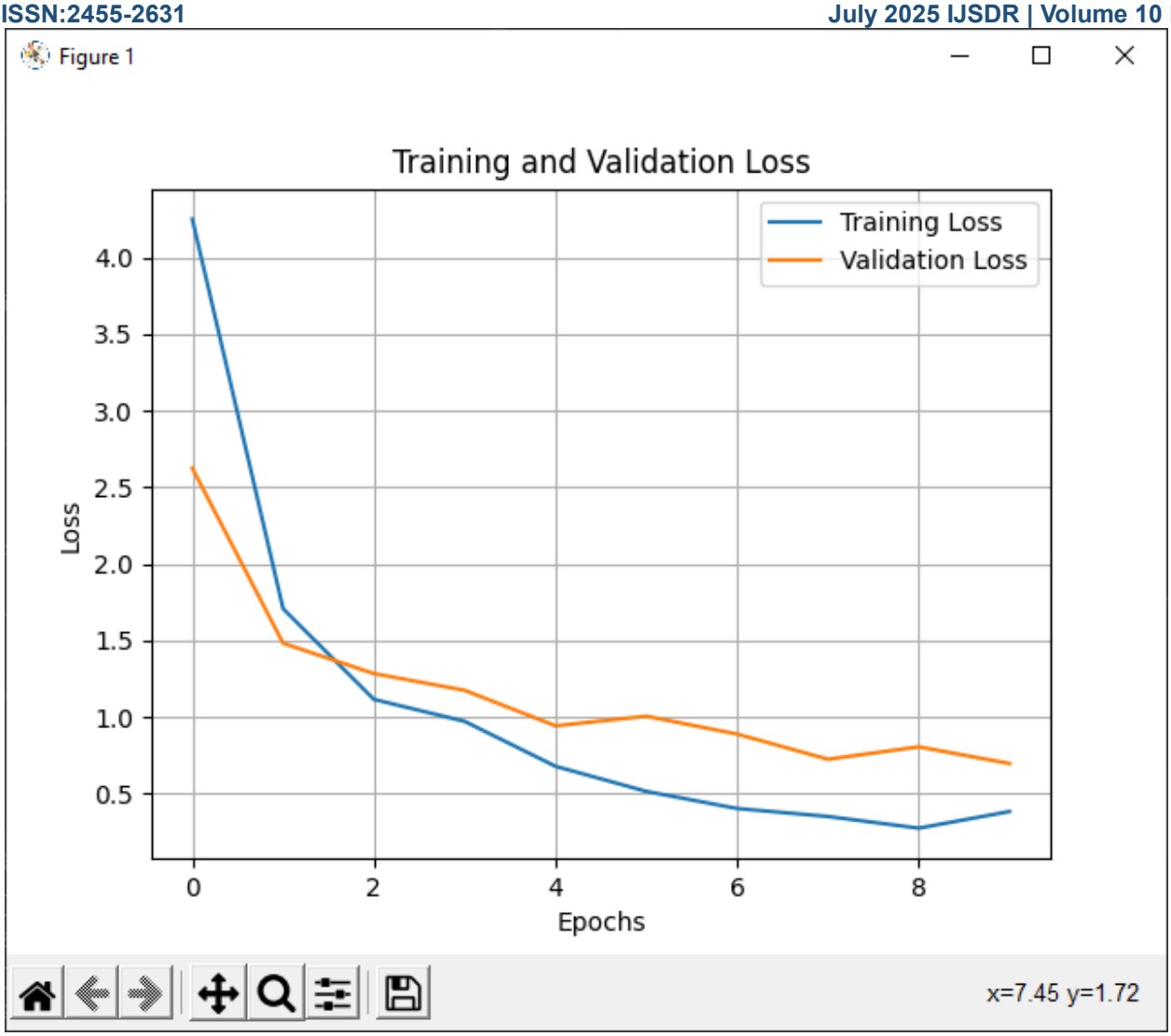
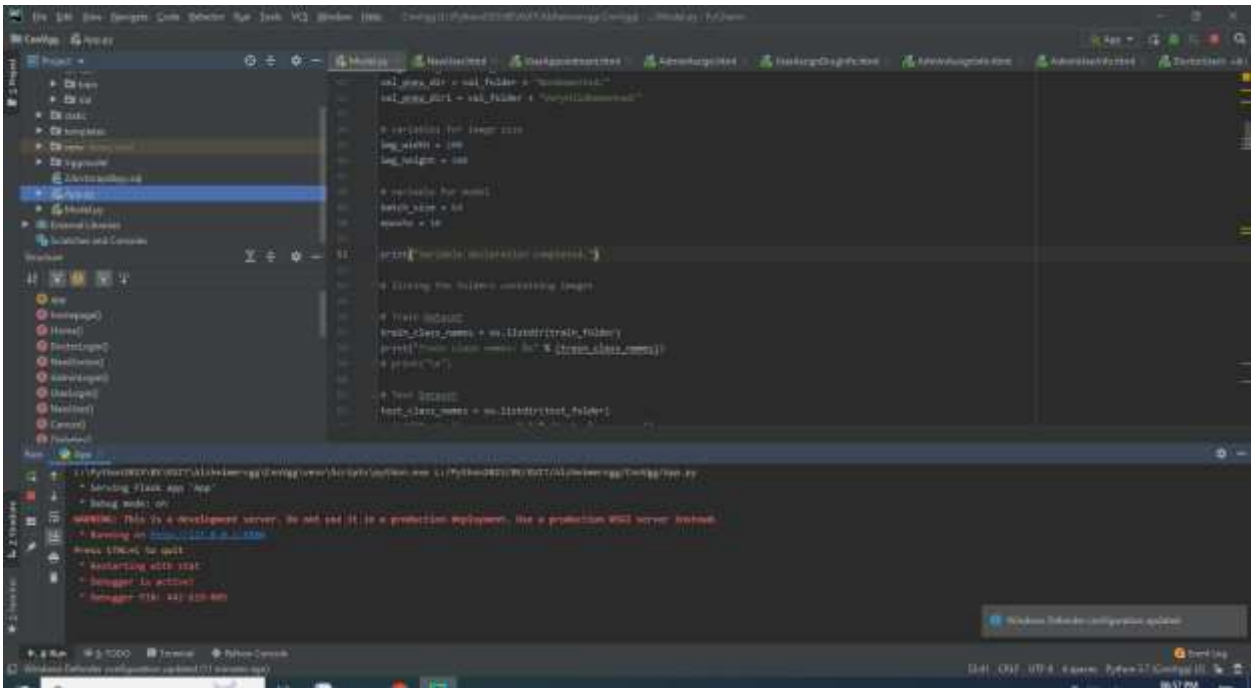
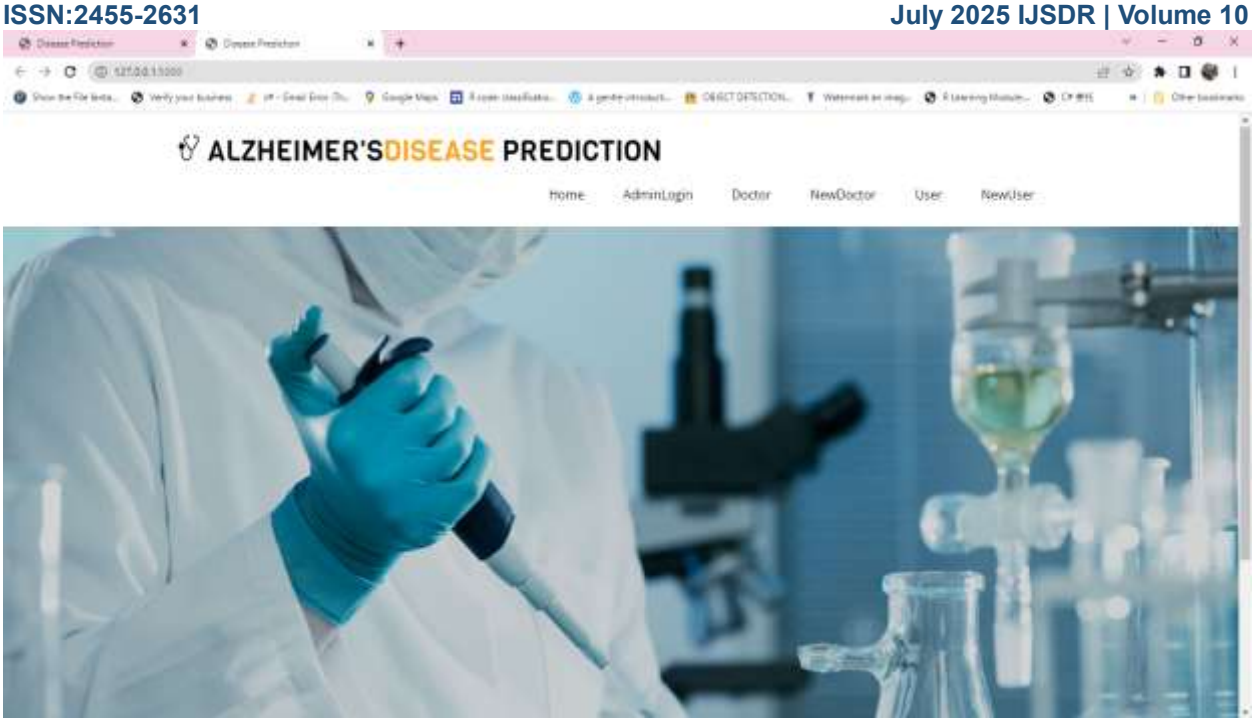
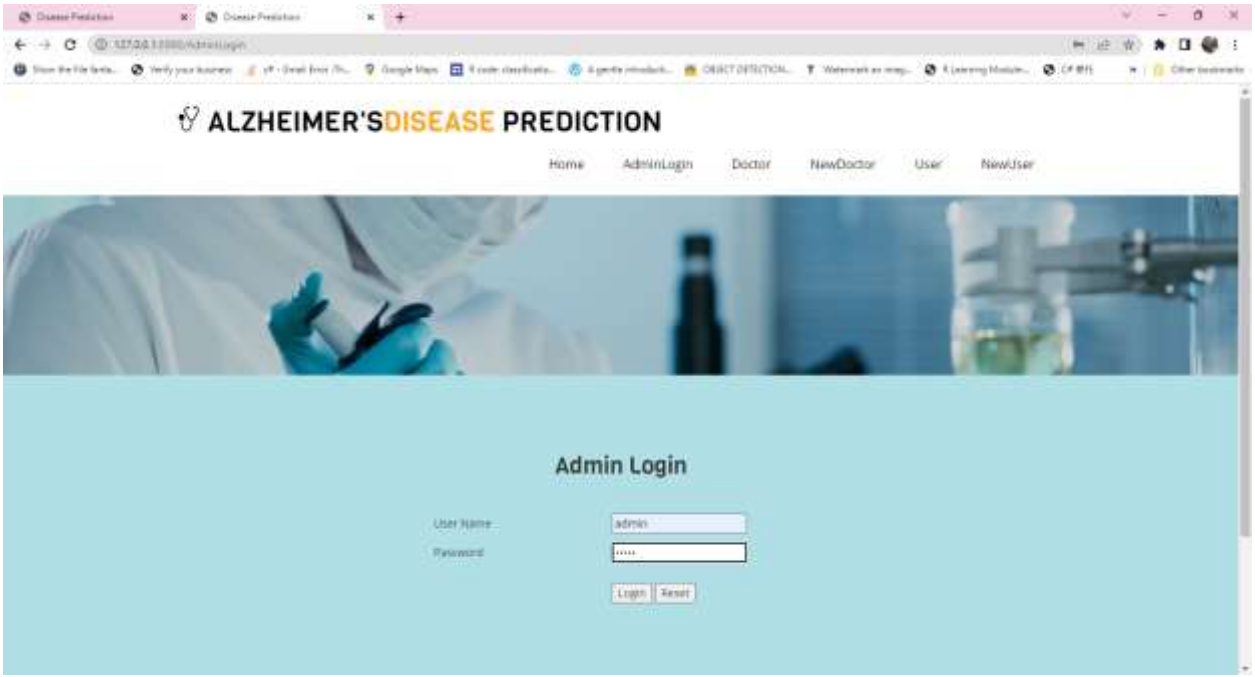
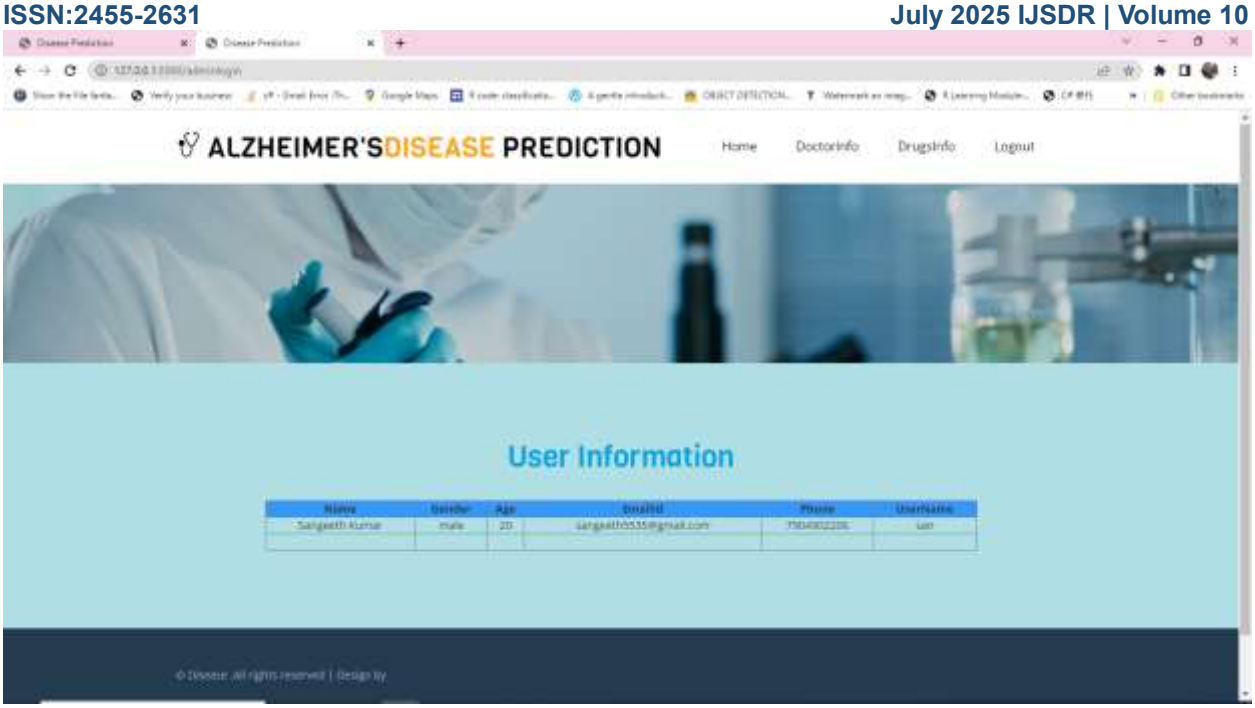


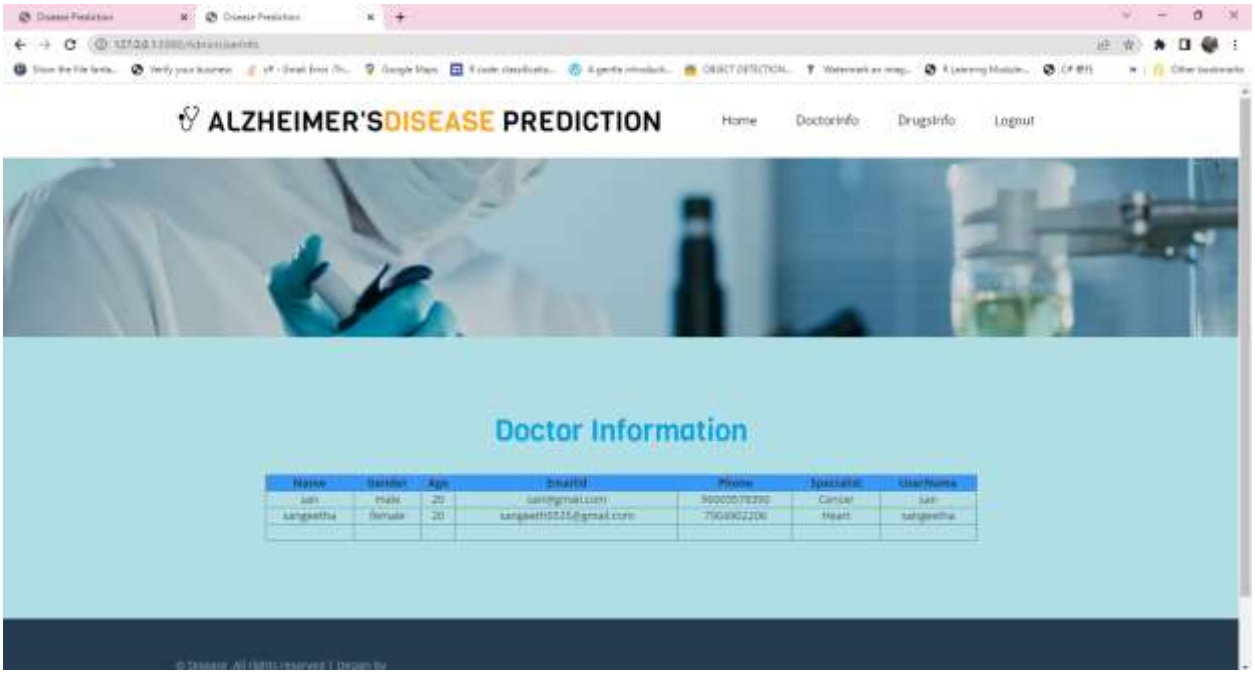
Figure 1
Training and Validation Loss
Training Loss
Validation Loss
Loss
Epochs
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8
x=7.45 y=1.72

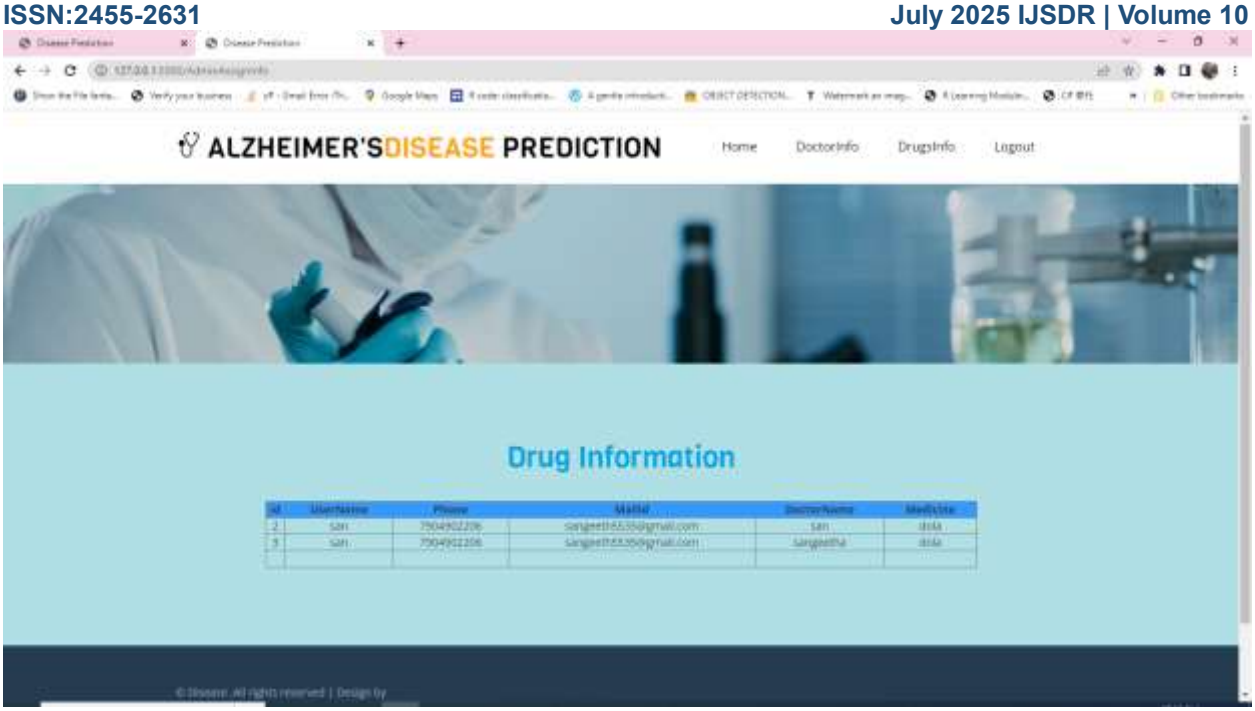


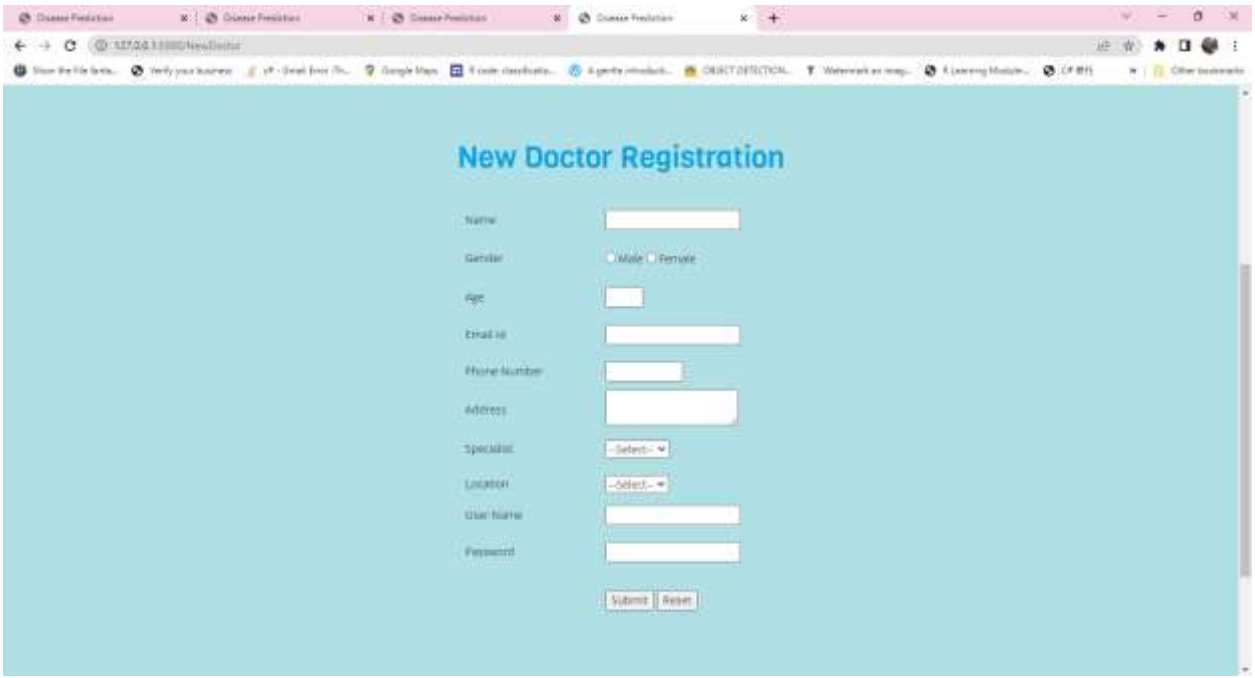

ALZHEIMER'SDISEASE PREDICTION
Home
AdminLogin
Doctor
NewDoctor
User
NewUser

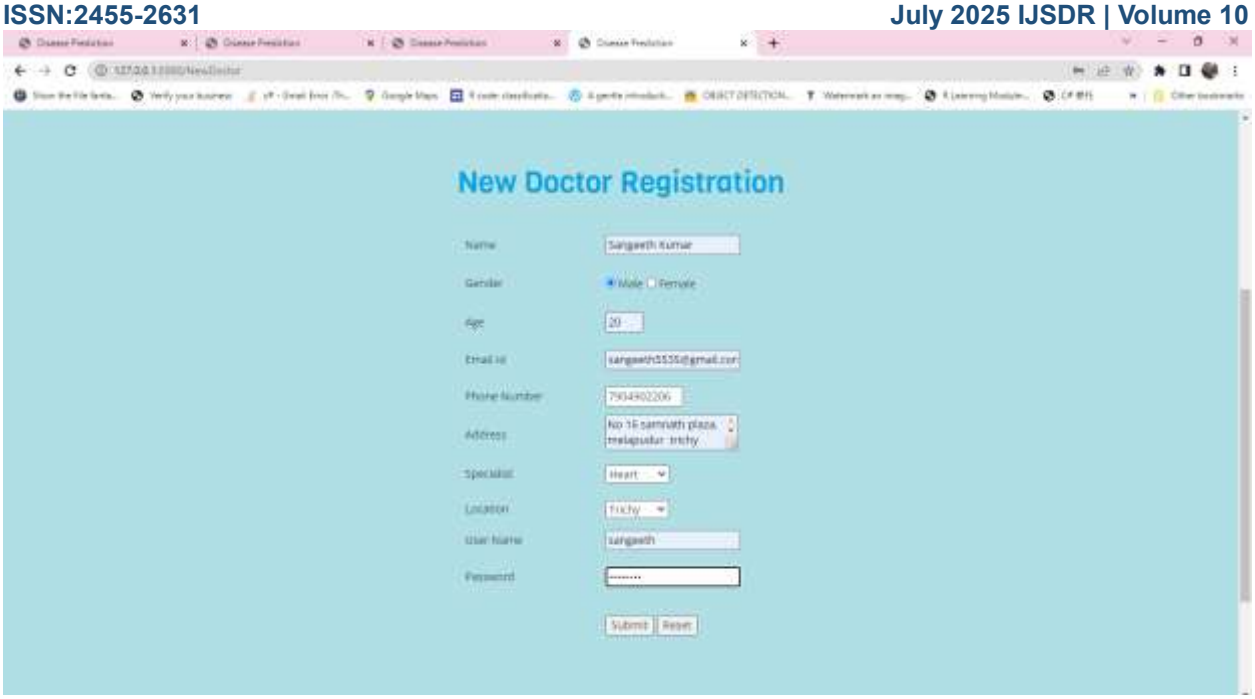

ALZHEIMER'SDISEASE PREDICTION
Home
AdminLogin
Doctor
NewDoctor
User
NewUser
Admin Login
User Name
admin
Password
Login
Reset


ALZHEIMER'SDISEASE PREDICTION
Home
DoctorInfo
DrugsInfo
Logout
User Information

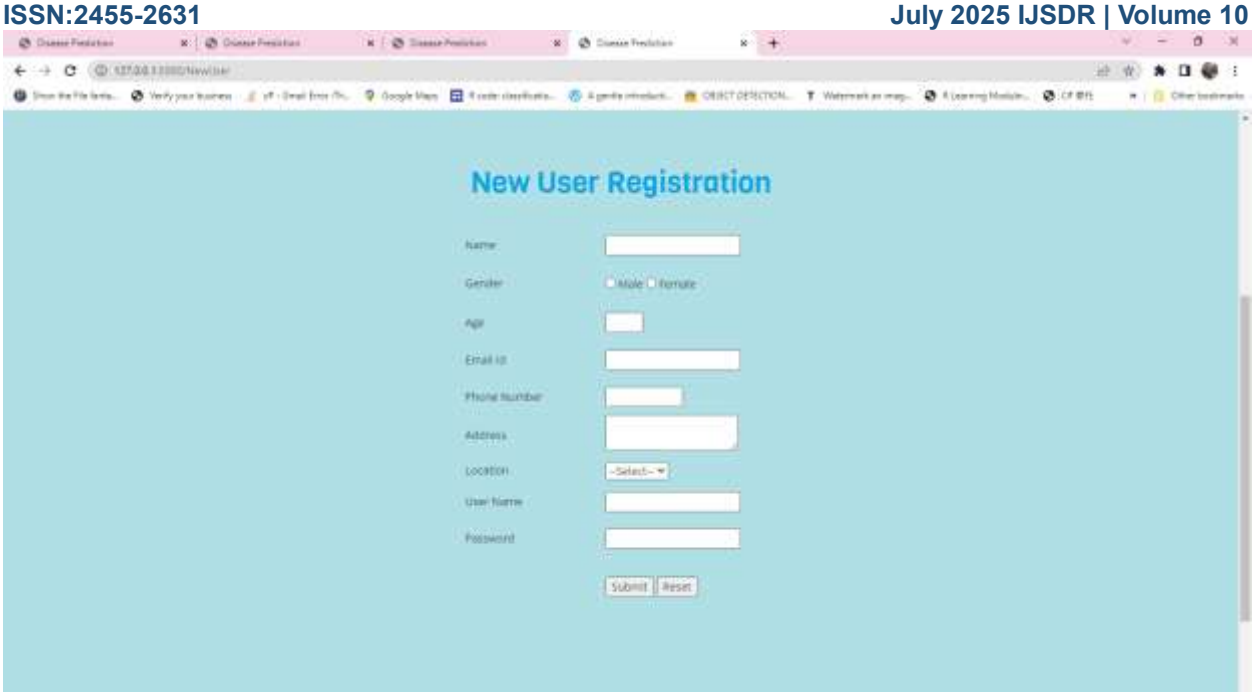

ALZHEIMER'SDISEASE PREDICTION
Home
DoctorInfo
DrugsInfo
Logout
Doctor Information

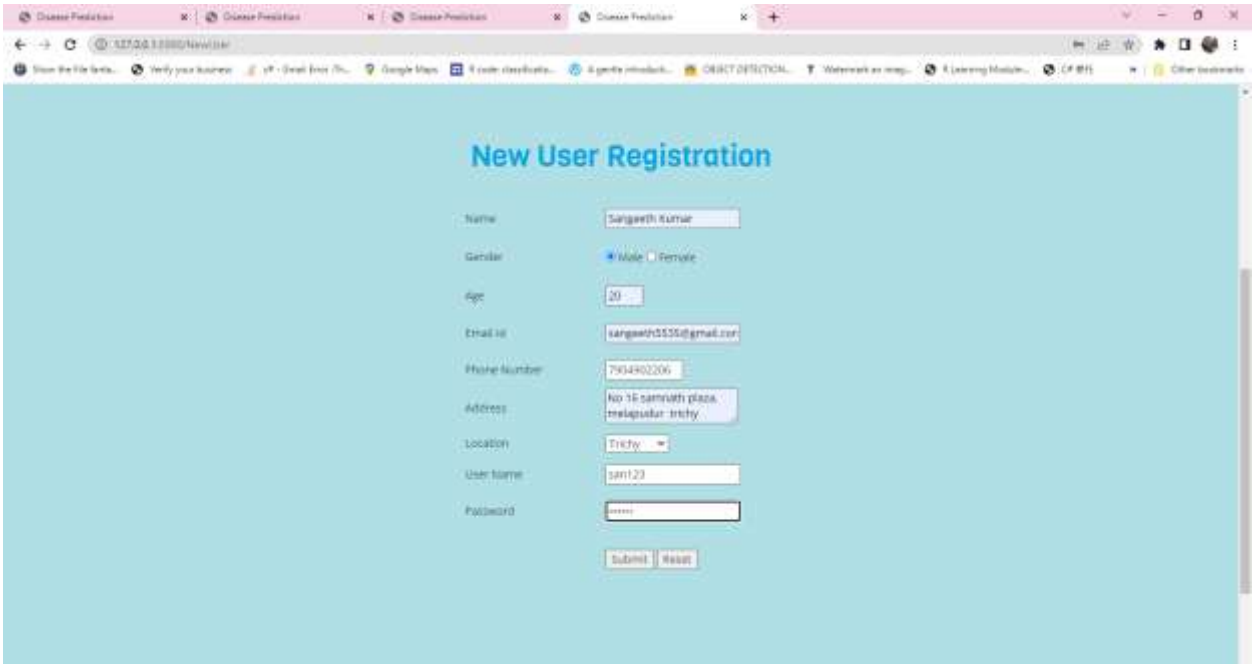

ALZHEIMER'SDISEASE PREDICTION
Home
DoctorInfo
DrugsInfo
Logout
Drug Information

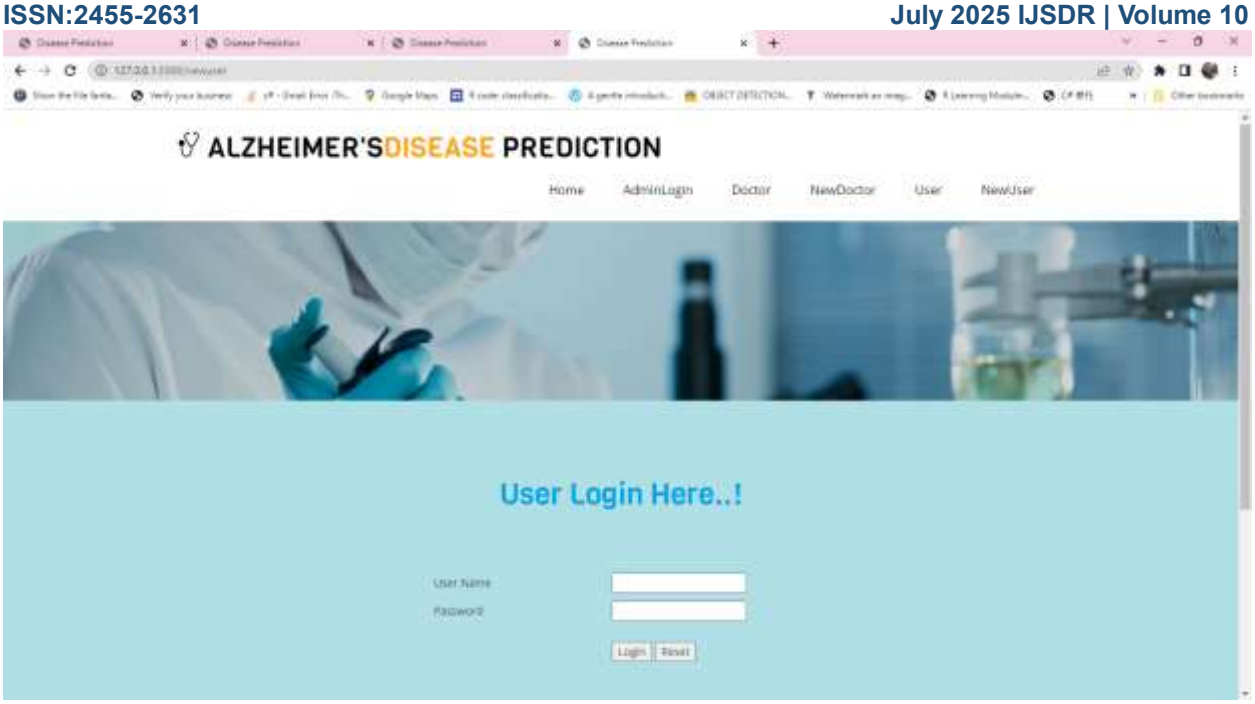

New Doctor Registration
Name
Gender
Male
Female
Age
Email id
Phone Number
Address
Specialist
Location
User Name
Password
Submit
Reset

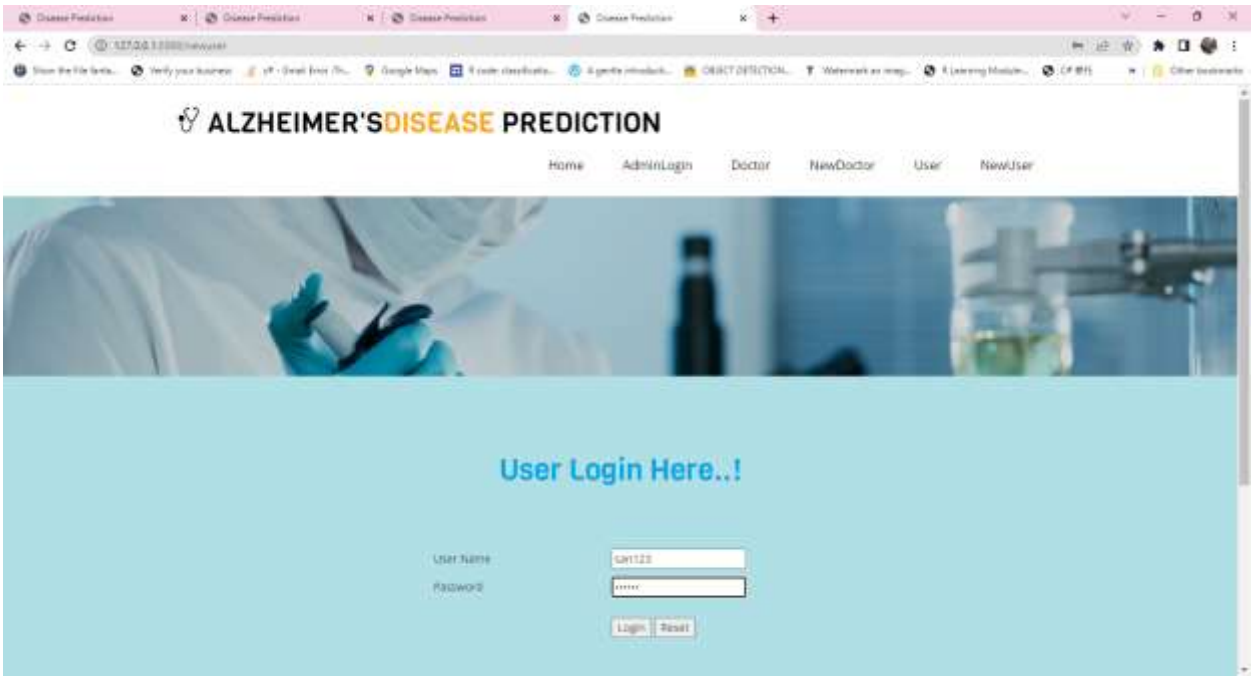

New Doctor Registration
Name
Sangeeth Kumar
Gender
Male
Female
Age
20
Email id
Phone Number
Address
No 16 samrudh plaza,
melapudur trichy
Specialist
Heart
Location
Trichy
User Name
sangeeth
Password
Submit
Reset

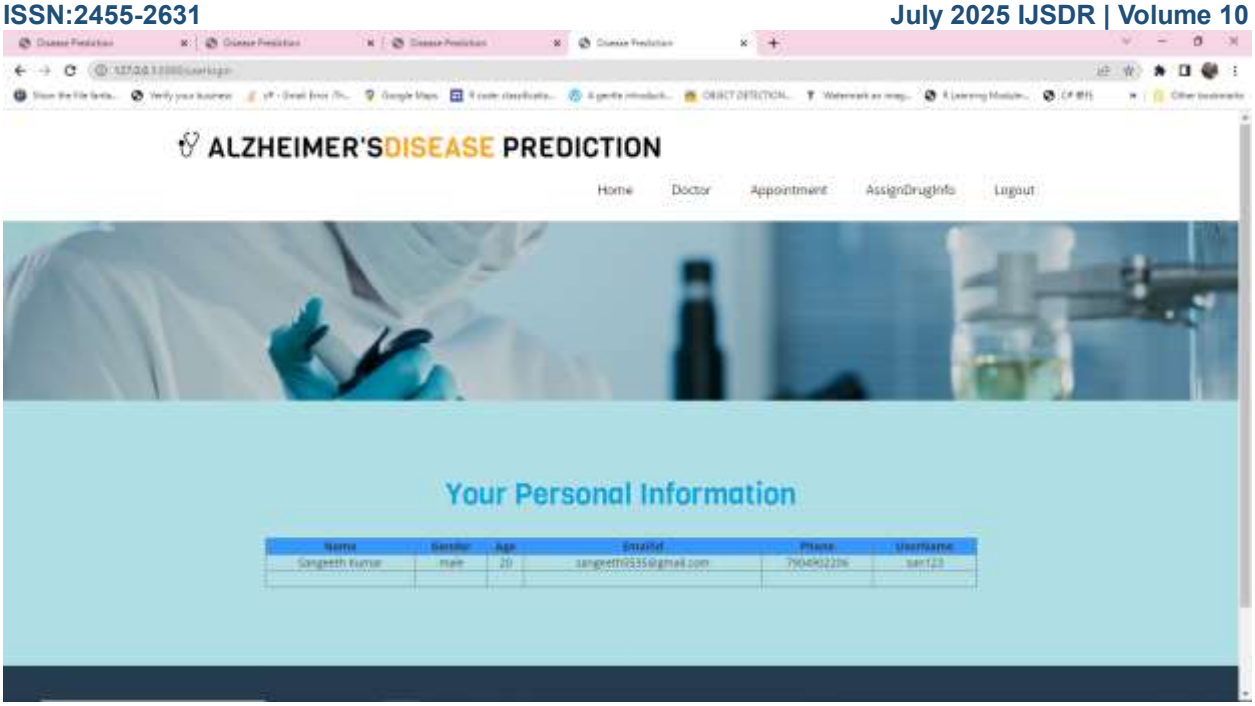

Record Saved
Go Back

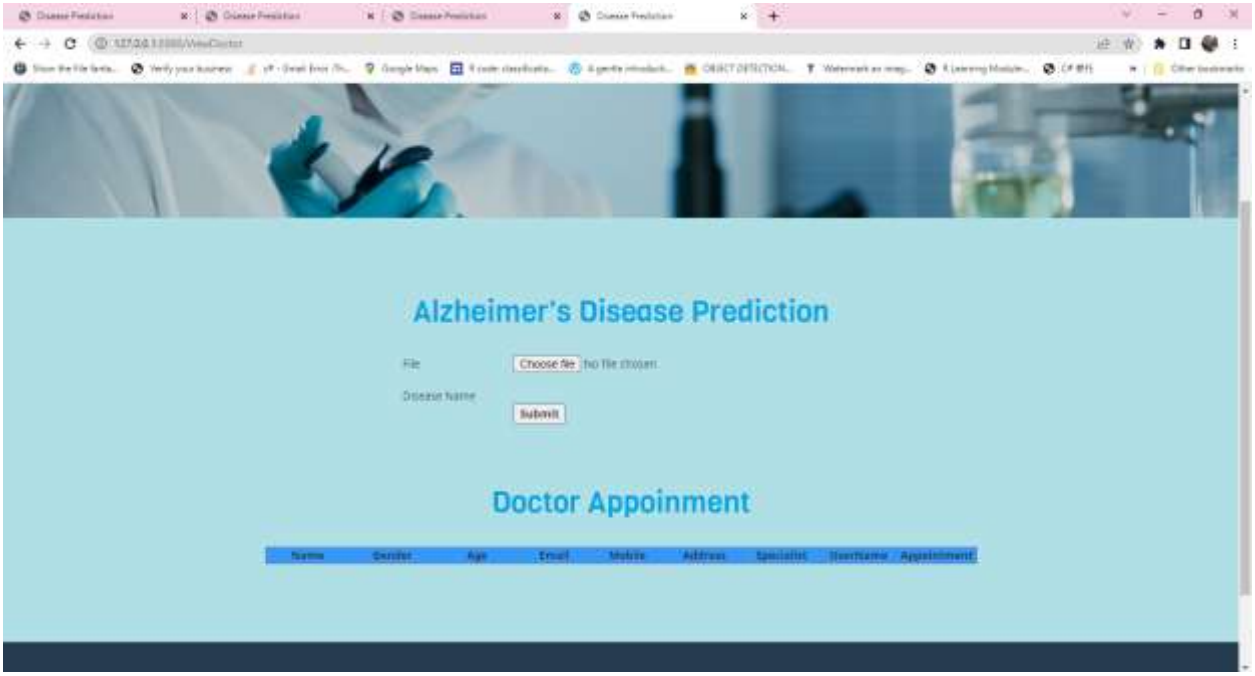

New User Registration
Name
Gender
Male Female
Age
Email id
Phone Number
Address
Location
--Select--
User Name
Password
Submit
Reset

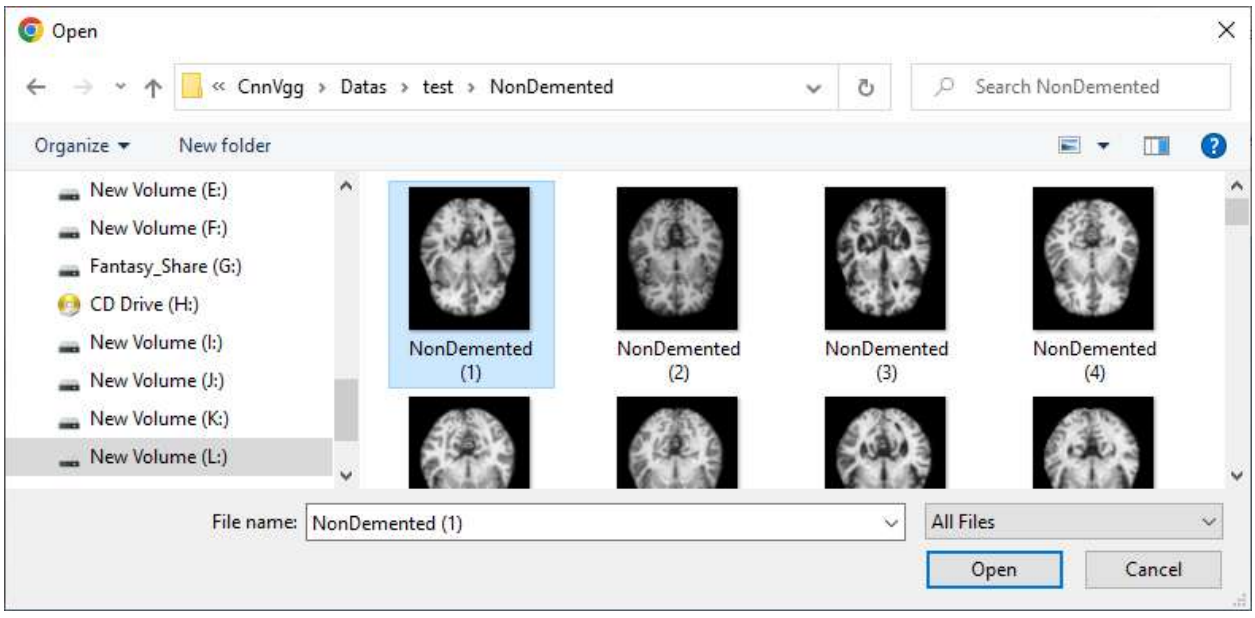

New User Registration
Name
Sangeeth Kumar
Gender
Male Female
Age
20
Email id
Phone Number
7904902206
Address
No 16 samrudh plaza
melapudur trichy
Location
Trichy
User Name
sam123
Password
Submit
Reset

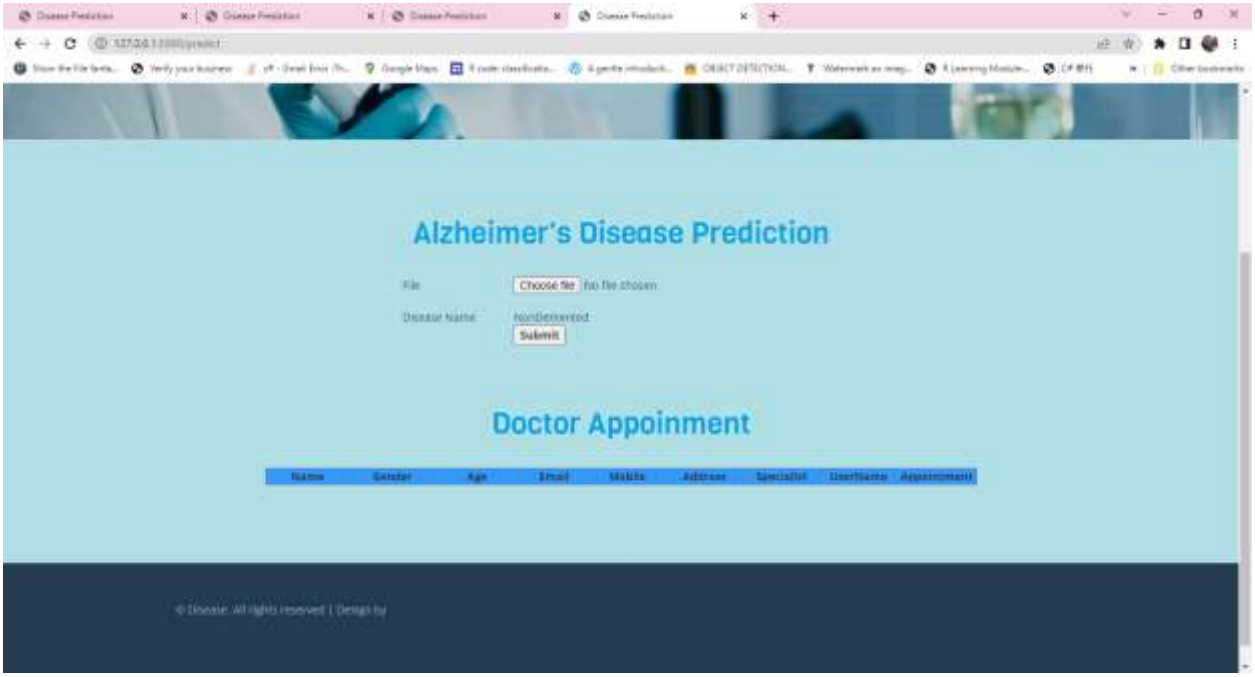

ALZHEIMER'SDISEASE PREDICTION
Home
AdminLogin
Doctor
NewDoctor
User
NewUser
User Login Here..!
User Name
Password
Login
Reset

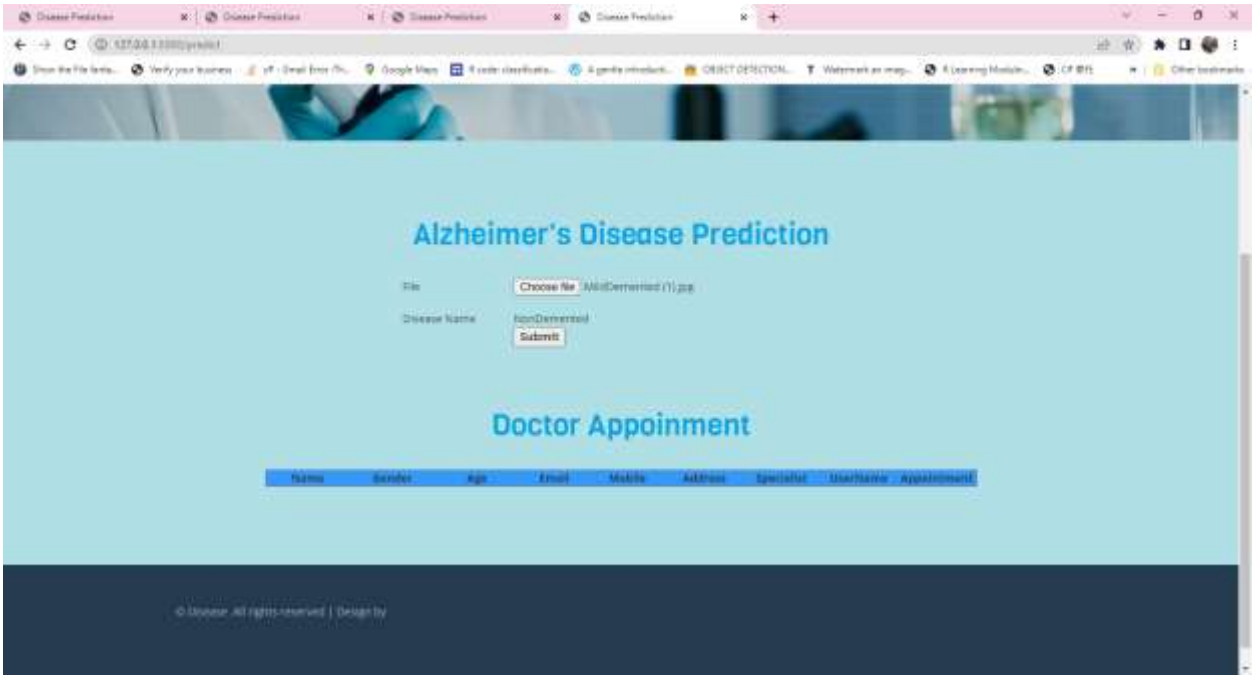

ALZHEIMER'SDISEASE PREDICTION
Home
AdminLogin
Doctor
NewDoctor
User
NewUser
User Login Here..!
User Name
Password
Login
Reset


ALZHEIMER'SDISEASE PREDICTION
Home
Doctor
Appointment
AssignDrugInfo
Logout
Your Personal Information


Alzheimer's Disease Prediction
File
Choose file
No file chosen
Disease Name
Submit
Doctor Appoinment

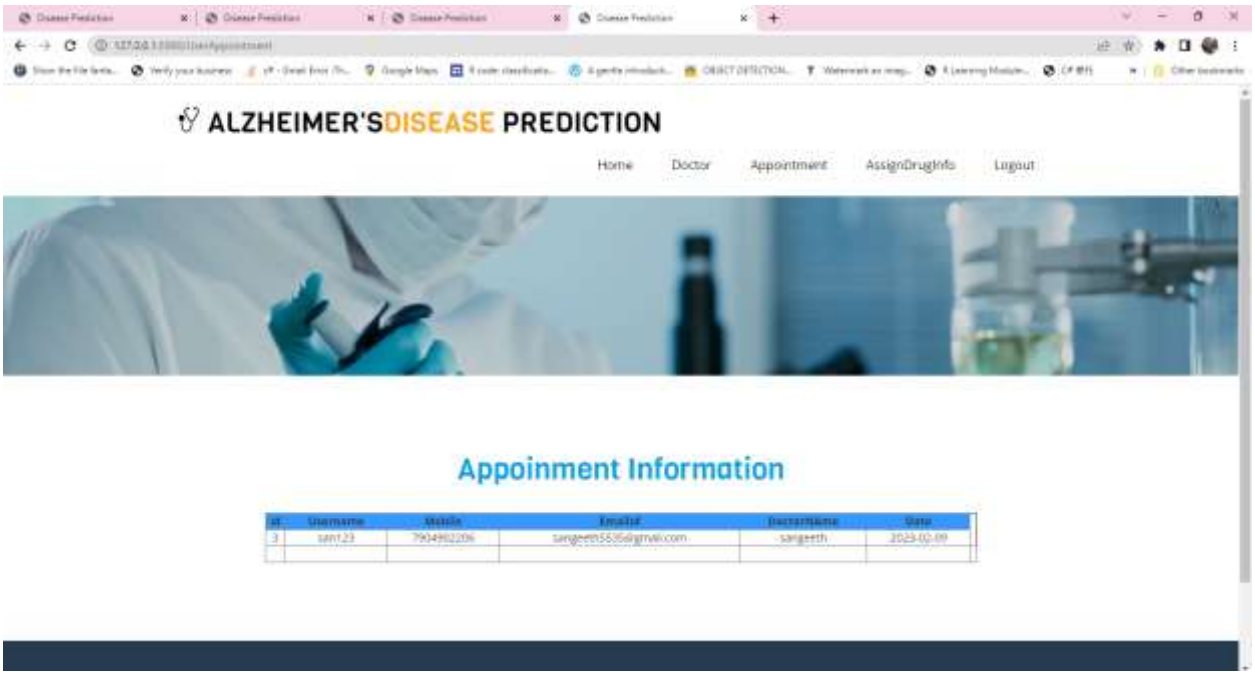

Open
CnnVgg
Datas
test
NonDemented
Search NonDemented
Organize
New folder
New Volume (E:)
New Volume (F:)
Fantasy_Share (G:)
CD Drive (H:)
New Volume (I:)
New Volume (J:)
New Volume (K:)
New Volume (L:)
NonDemented (1)
NonDemented (2)
NonDemented (3)
NonDemented (4)
File name:
NonDemented (1)
All Files
Open
Cancel

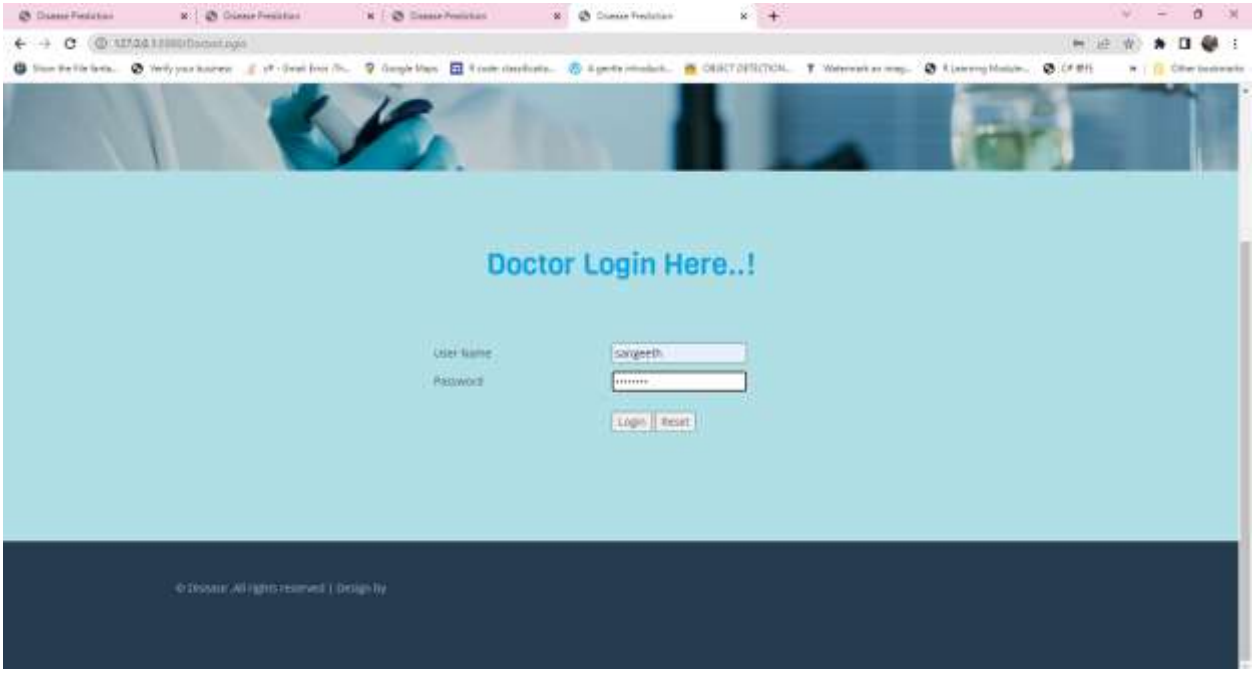

Alzheimer's Disease Prediction
File
Choose file
No file chosen
Disease Name
nonDemented
Submit
Doctor Appoinment

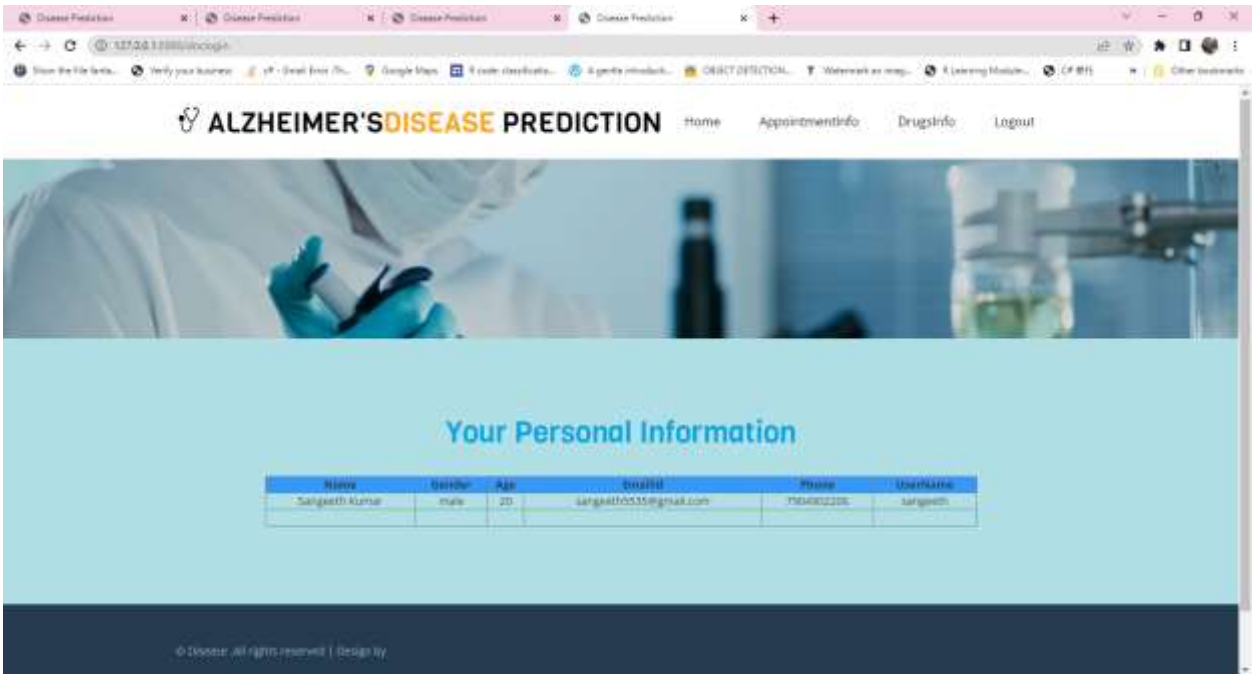

Alzheimer's Disease Prediction
File
Choose file
Disease Name
nonDemented
Submit
Doctor Appoinment

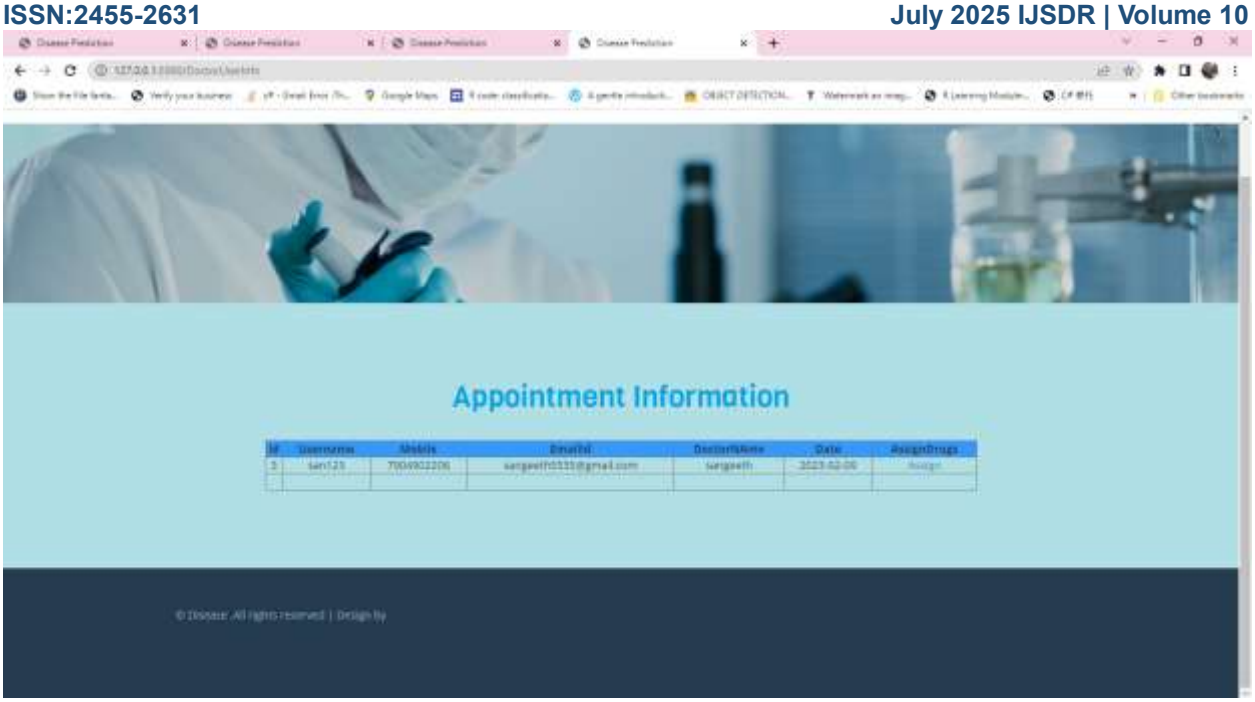

Submit
Doctor Appoinment
Name
Gender
Age
Email
Mobile
Address
Specialist
UserName
Appointment
sathi
male
20
sanki@gmail.com
9600557000
ssn
Cancer
sathi
Appointment
sangeetha
female
20
sangeeth5535@gmail.com
7904902206
no 16 samnath plaza melapudur trichy no 16 samnath plaza melapudur trichy
Heart
sangeetha
Appointment
Sangeeth Kumar
male
20
sangeeth5535@gmail.com
7904902206
no 16 samnath plaza melapudur trichy no 16 samnath plaza melapudur trichy
Heart
sangeeth
Appointment

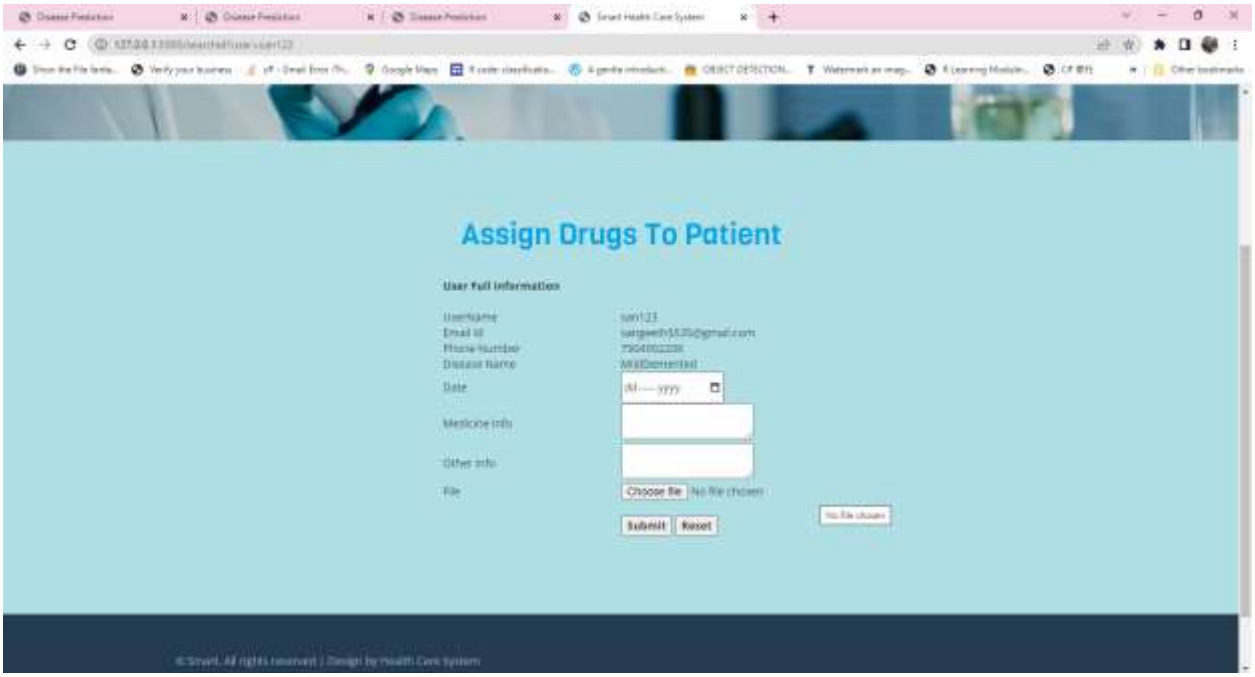

Record Saved
Go Back

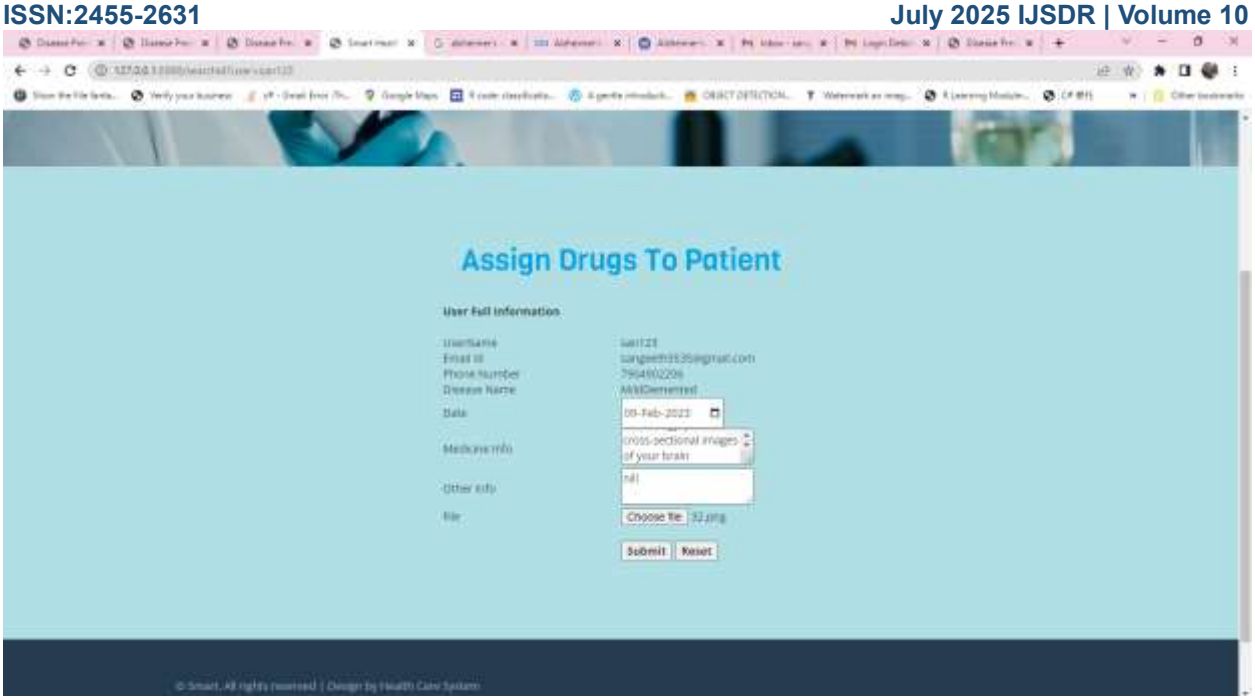

ALZHEIMER'SDISEASE PREDICTION
Home
Doctor
Appointment
AssignDrugInfo
Logout
Appoinment Information
Id
Username
Mobile
Email
DoctorName
Date
3
sam123
7904902206
sangeeth5535@gmail.com
sangeeth
2023-02-09

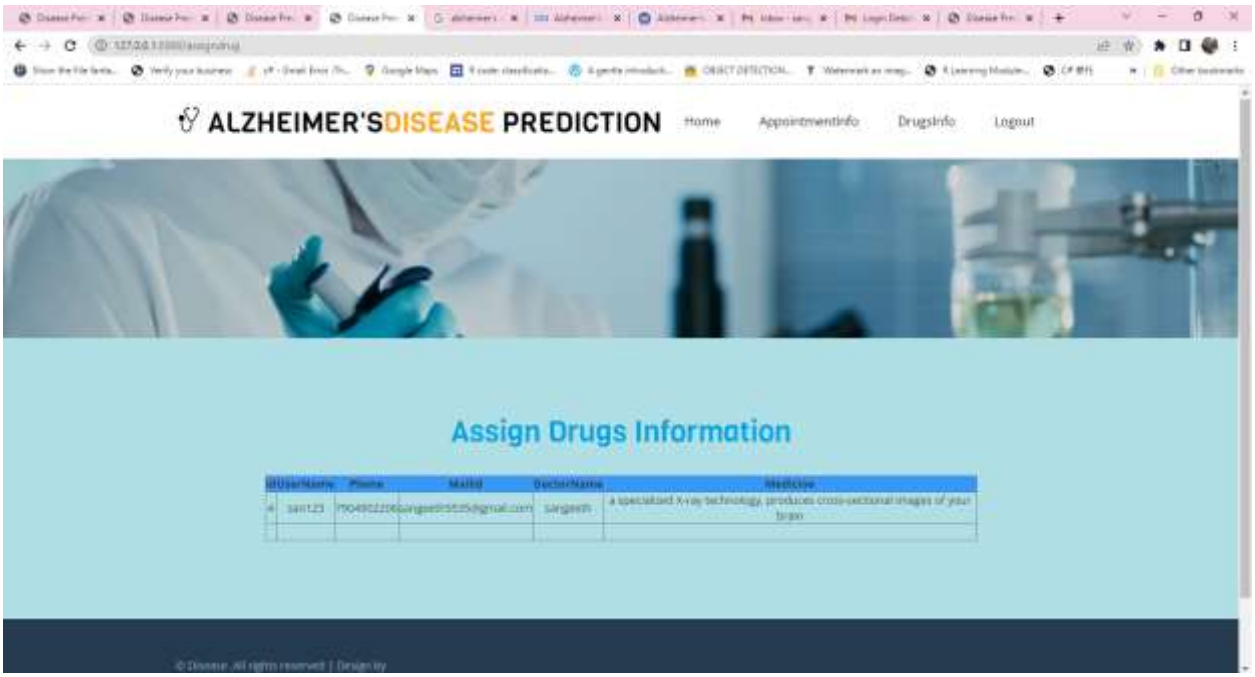

Doctor Login Here..!
User Name
sangeeth
Password
Login
Reset

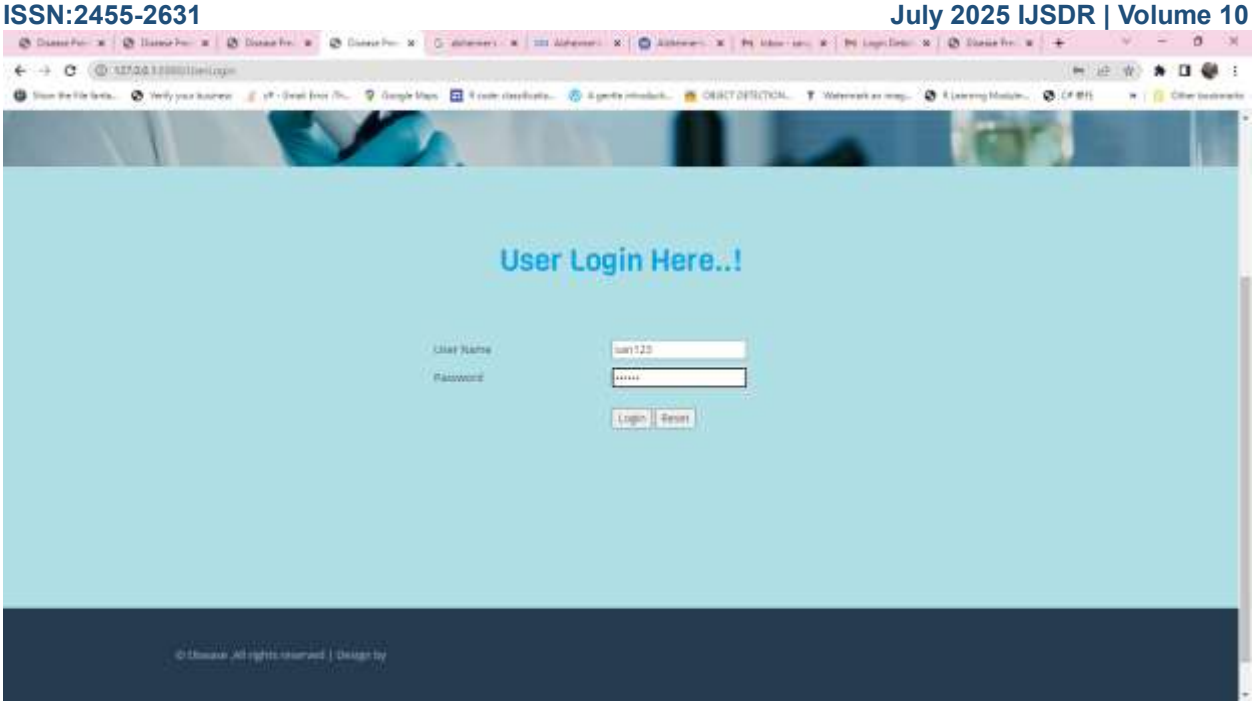

ALZHEIMER'SDISEASE PREDICTION
Home
AppointmentInfo
DrugsInfo
Logout
Your Personal Information
Name
Gender
Age
Email Id
Phone
UserName
Sangeeth Kumar
male
20
sangeeth0555@gmail.com
sangeeth

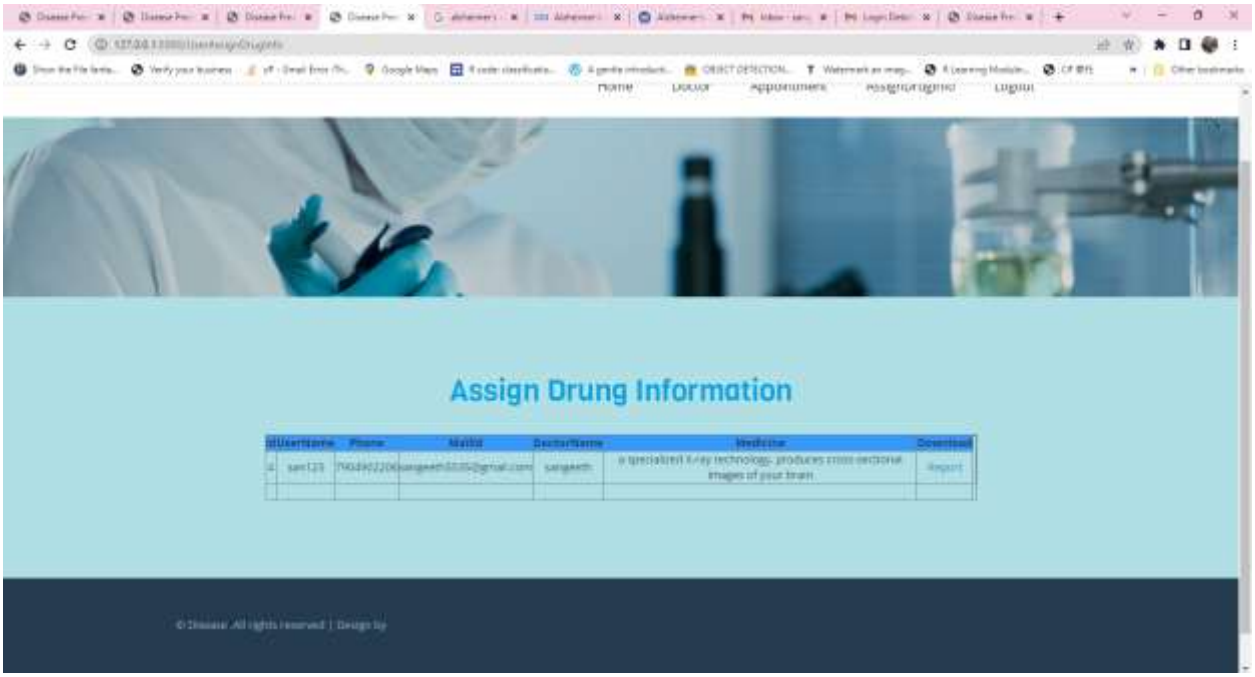

Appointment Information

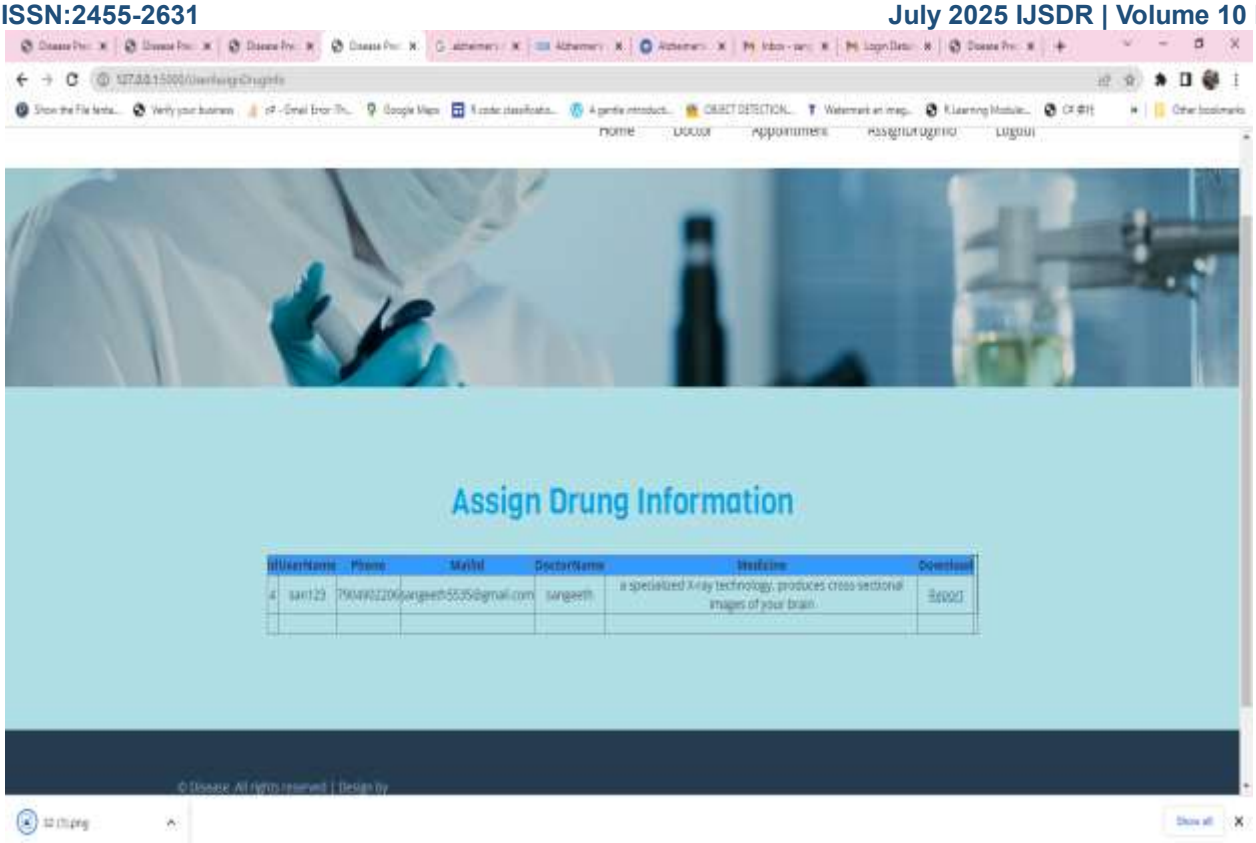

Assign Drugs To Patient
User Full Information
Submit
Reset


Assign Drugs To Patient
User Full Information
Submit
Reset


ALZHEIMER'SDISEASE PREDICTION
Home
AppointmentInfo
DrugsInfo
Logout
Assign Drugs Information


User Login Here..!
User Name
Password
Login
Reset


Home
Doctor
Appointment
Assign Drug Info
Logout
Assign Drung Information



REFERENCES

[1] Basher, Abol, et al. "Volumetric feature-based Alzheimer's disease diagnosis from sMRI data using a convolutional neural network and a deep neural network." IEEE Access 9 (2021): 29870-29882.

[2] Abrol, Anees, et al. "Deep residual learning for neuroimaging: an application to predict progression to Alzheimer's disease." Journal of neuroscience methods 339 (2020): 108701.

[3] De Silva, S., S. Dayarathna, and D. Meedeniya. "Alzheimer's Disease Diagnosis using Functional and Structural Neuroimaging Modalities." Enabling Technology for Neurodevelopmental Disorders. Routledge, 2021. 162-183.

[4] Garcia-Gutierrez, Fernando, et al. "Diagnosis of Alzheimer's disease and behavioural variant frontotemporal dementia with machine learning-aided neuropsychological assessment using feature engineering and genetic algorithms." International Journal of Geriatric Psychiatry 37.2 (2022).

[5] Zhang, Jie, et al. "A 3D densely connected convolution neural network with connection-wise attention mechanism for Alzheimer's disease classification." Magnetic Resonance Imaging 78 (2021): 119-126.

[6] Liu, Manhua, et al. "A multi-model deep convolutional neural network for automatic hippocampus segmentation and classification in Alzheimer's disease." Neuroimage 208 (2020): 116459.

[7] Mendoza-Léon, Ricardo, et al. "Single-slice Alzheimer's disease classification and disease regional analysis with Supervised Switching Autoencoders." Computers in biology and medicine 116 (2020): 103527.

[8] Kumar, L. Sathish, et al. "AlexNet approach for early-stage Alzheimer's disease detection from MRI brain images." Materials Today: Proceedings 51 (2022): 58-65.

[9] Shaji, Sreelakshmi, Nagarajan Ganapathy, and Ramakrishnan Swaminathan. "Classification of alzheimer condition using MR brain images and inception-residual network model." Current Directions in Biomedical Engineering 7.2 (2021): 763-766.

[10] Xu, Zelin, et al. "Diagnosis of Alzheimer's Disease Based on the Modified Tresnet." Electronics 10.16 (2021): 1908.