DETECTION OF ALZHEIMER'S DISEASE USING MAGNETIC RESONANCE IMAGES BASED ON MACHINE LEARNING METHOD

A Project Report

submitted in partial fulfillment of the requirements

for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

by

CH. Manoj Virinchi	**18B81A05E3**
G. Sathvika	**18B81A05G4**
M. Shiva Nandh Reddy	**18B81A05G6**

Under the supervision of

Dr M. Sridevi, Associate Professor

Department of Computer Science and Engineering

CVR COLLEGE OF ENGINEERING

(An Autonomous institution, NBA, NAAC Accredited and Affiliated to JNTUH, Hyderabad)

Vastunagar, Mangalpalli (V), Ibrahimpatnam (M), Rangareddy (D), Telangana- 501 510

April 2022

DECLARATION

I certify that

a. The work contained in this report is original and has been done by me under the guidance of my supervisor(s).

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed the guidelines provided by the Institute in preparing the report.

d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Place: **Signature of the Student**

Date: **Roll No.**

CERTIFICATE

This is to certify that the project report entitled "**DETECTION OF ALZHEIMER'S DISEASE USING MAGNETIC RESONANCE IMAGES BASED ON MACHINE LEARNING METHOD"** submitted by **Mr. CH. Manoj Virinchi(18B81A05E3), Ms. G. Sathvika(18B81A05G4), Mr. M. Shiva Nandh Reddy(18B81A05G6)** to the CVR College of Engineering in partial fulfillment of the requirements for the award of Bachelor of Technology in **Computer Science and Engineering** is a bonafide record of work carried out by him/her under my/our guidance and supervision during Academic Year 2021-22.

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

Supervisor **Head of the Department**

Dr M. Sridevi **Dr. A. Vani Vathsala**

Associate Professor

Date:

External Examiner

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We avail this opportunity to express our deep sense of gratitude and hearty thanks to management of CVR College of Engineering, for providing congenial atmosphere and encouragement.

We would like to thank **Mrs. A. VANI VATHSALA Head of the Department, Computer Science and Engineering** for her expert guidance and encouragement at various levels of our Project.

We are thankful to our guide **Mr. S. SRINIVAS, Assistant professor** for his sustained inspiring Guidance and cooperation throughout the process of this project. His wise counsel and suggestions were invaluable.

We express our deep sense of gratitude and thanks to all the **Teaching** and **Non-Teaching Staff** of our college who stood with us during the project and helped us to make it a successful venture.

CH. MANOJ VIRINCHI (18B81A05E3)
G. SATHVIKA (18B81A05G4)
M. SHIVA NANDH REDDY (18B81A05G6)

ABSTRACT

Alzheimer's disease is an unrepairable degenerative brain disease. Every four seconds, someone in the world is diagnosed with Alzheimer's disease. It is a neurodegenerative disorder that affects elderly people. The leading cause of dementia is Alzheimer's disease. Dementia causes a reduction in reasoning abilities and interpersonal coping skills, which affects people's ability to function independently. The patient will forget recent events in the early stages. If the illness progresses, they will gradually forget whole events. This is a progressive disease and early detection and classification of AD can majorly help in controlling the disease. Recent studies use voxel-based brain MR image feature extraction techniques along with machine learning algorithms for this purpose. Grey and white matter of the brain gets affected and damaged due to AD and so studying these both prove to be more effective in predicting the disease. It's crucial to catch the disease early on. This paper proposes a model that takes brain MRI sample images as input and determines whether a person has mild, moderate, or no Alzheimer's disease as an output. We are using the VGG19 and DenseNet169 architectures for this classification, providing a comparative analysis of which architecture shows promising results.

TABLE OF CONTENTS

Table of Contents

LIST OF FIGURES

LIST OF ABBREVIATIONS

AD	Alzheimer's Disease
MRI	Magnetic Resonance Imaging
MCI	Mild Cognitive Impairment
CSF	Cerebrospinal Fluid
MTL	Media Temporal Lobe
PCA	Principal Component Analysis
ML	Machine Learning
NN	Neural Network
CNN	Convolutional Neural Networks
DNN	Deep Neural Network
RNN	Recurrent Neural Network
SVM	Support Vector Machine
KNN	K-Nearest Neighbor
ResNet	Residual Neural Network
VGG	Visual Geometry Group
DenseNet	Dense Convolutional Network

CHAPTER-1
INTRODUCTION

1.1 Motivation

Alzheimer's disease (AD) is the leading cause of dementia in older adults. There is currently a lot of interest in applying machine learning to find out metabolic diseases like Alzheimer's and Diabetes that affect a large population of people around the world. Their incidence rates are increasing at an alarming rate every year. In Alzheimer's disease, the brain is affected by neurodegenerative changes. As our aging population increases, more and more individuals, their families, and healthcare will experience diseases that affect memory and functioning. These effects will be profound on the social, financial, and economic fronts. In its early stages, Alzheimer's disease is hard to predict. A treatment given at an early stage of AD is more effective, and it causes fewer minor damage than a treatment done at a later stage. Several techniques such as Decision Tree, Random Forest, Support Vector Machine, Gradient Boosting, and Voting classifiers have already been employed to identify the best parameters for Alzheimer's disease prediction. High-dimensional classification methods have been commonly used to explore Magnetic Resonance Imaging (MRI) for automatic classification of neurodegenerative diseases like AD and MCI. Early AD or MCI can be diagnosed through proper examination of several brain biomarkers such as Cerebrospinal Fluid (CSF), Media Temporal Lobe atrophy (MTL) and so on. Abnormal concentrations of the mentioned biomarkers on MRI images can be a potential sign of AD or MCI. The existing classification scheme can be used by clinicians to make diagnoses of these diseases. Therefore, classification of AD is usually a very daunting and time-consuming task. To improve the accuracy of the model CNN models like VGG16, VGG19, ResNet etc., have been used. But their accuracy is still struggling at 90%. To upgrade the accuracy furthermore a robust CNN model DenseNet169 has been proposed. These techniques have been implemented mostly from scratch, making it really difficult to achieve any meaningful result within a short span of time. Hence, we trained high dimensional Deep Neural Network (DNN) models in order to achieve meaningful results very quickly.

1.2 Problem Statement

Diagnosis of Alzheimer's disease (AD) is often difficult, especially early in the disease process at the stage of mild cognitive impairment (MCI). Alzheimer's disease is the most common form of dementia and may contribute to 60-70% of cases. It has been proved that early diagnosis is key to promoting early and optimal management. Yet, it is at this stage that treatment is most likely to be effective, so there would be great advantages in improving the diagnosis process. Diagnosing Alzheimer's is also an expensive and rigorous process as the patient needs to go through several number of procedures which are costly as well as time taking. In the existing system it is difficult to identify if a person is suffering from Alzheimer's disease. It can only be done by clinical history and by knowing if the person has some genetic disorder. Sometimes it is also possible that the doctor may not be able to detect the disease. To tackle the above problems and to diagnose Alzheimer's at a very early-stage to reduce the severity of the disease and help the patients to improve the quality of life, Machine learning approach has been introduced. These machine learning models not only help the patients overcome those problems, they also help in providing more accurate results with less error.

1.3 Project Objectives

Alzheimer's Disease (AD) is a neurodegenerative disease affects primarily the elderly population. It is a progressive disease and the fact that there is no treatment to stop or reverse the progression of the disease. The brain is considered one of the most crucial organs in our body. All the activities and responses that allow us to think and believe are controlled and facilitated by the brain. It also empowers our sentiments and recollections. Alzheimer's disease is brain dysfunction which is unrepairable and progressive in nature. Someone in the world is diagnosed with Alzheimer's disease every four seconds. It enhances at a languid pace and tears down the memory cells, thereby destroying an individual's thinking ability. It's a degenerative nerve disorder that leads to loss of function or even death of neurons. The average life expectancy after an Alzheimer's diagnosis is only about four to eight years. On an average, 1 out of 10 people over the age of 65 is affected by this condition, but sometimes it can strike at a younger age and has been diagnosed in several people in their 20s.

According to the reports [1] from 2005 through 2030, there is a steady growth in the percentage estimate of the number of people affected by AD. Presently 40 million people suffer from AD worldwide. It is distinctly possible to reach 135 million by 2050 [2]. However, an interesting feature of AD is, though incurable, early detection and appropriate treatment of the disease can control the degeneration of neurons. In the current context, Computer-Aided Diagnostics uses advanced computer programs and algorithms in the field of image processing and pattern recognition for identification of Features of Interest or Region of Interest (FOI / ROI) in the MR image under observation. The developed programs are expected to highlight the necessary features while keeping a control on the false negative rate systems when carefully developed are much better inaccuracies and can greatly assist the neurologist to understand the physiological changes in the brain. It is, for this reason, a significant amount of research is underway across the globe towards the classification and detection of different stages of neurodegenerative diseases including Alzheimer's disease.

Machine learning is used to interpret and analyze data. Furthermore, it can classify patterns and model data. It permits decisions to be made that could not be made generally utilizing routine systems while sparing time and endeavors. Machine learning methodologies have been extensively used for computer-aided diagnosis in medical image formation mining and retrieval with wide variety of other applications especially in detection and classifications of brain disease using MRI images and x-rays. It has just been generally late that AD specialists have endeavored to apply machine learning towards AD prediction. It is essential to diagnose the disease as soon as possible. This project proposes a model that takes brain MRI sample images as input and determines whether a person has mild, moderate, or no Alzheimer's disease as an output. We are using the VGG19 and DenseNet169 architectures for this classification, providing a comparative analysis of which architecture shows promising results.

The main objective of the project is to make the diagnosis of the disease easier, to detect the disease in its early stages and use the machine learning algorithm efficiently.

1.4 Project Report Organization

Chapter 1: Introduction

This chapter is the introduction to this project report. This chapter gives the basic idea of the project. It deals with things such as the motivation behind this project, problem statement of the project, the objectives of the. This Chapter also consists of Project report organization which gives the idea about what each chapter in this project report does consists of.

Chapter 2: Literature Survey

This chapter consists of the details about characteristics of the problem and also the different types of design challenges which are required to design the project. This chapter also contains the proposed model for the project.

Chapter 3: Software & Hardware specifications

This chapter consists of different types of requirements which are required to build the project. This chapter deals with the requirements like software requirements, hardware requirements, Functional requirements and also non-functional requirements. It also discusses about system specifications such as hardware and software specifications.

Chapter 4: Proposed System Design

This chapter consists of different types of diagrams which depicts the diagrammatical representation of the project. This chapter consists of diagrams like use case diagram, Class diagram, Activity diagram, Sequence diagram. This also consists of system architecture and technology description.

Chapter 5: Implementation and Testing

This fifth chapter discuss about how the project is implemented. This also contains the details about the implementation environment which provides information about the platform which the project is implemented.

Chapter 6: Conclusion and future scope

This is the last chapter which is about conclusion of the report and consists of future scope of the project. Future Scope explains how this project can be used in the future and how it can be developed from existing version.

Chapter 7: References

This consists of all the references used in the development of the project.

CHAPTER-2
LITERATURE SURVEY

2.1 Existing work

Over the years, several researchers have used different approaches to diagnose Alzheimer's Disease. All researchers of this domain focus on monitoring a patient's health change, clinical progression of disease, and reaction to the therapy. It is most challenging for them to find relevant bio-markers that represent AD and MCI well. Their goal is not only to diagnose this disease at an early stage, but also identify which people are most likely to develop AD. Magnetic Resonance Imaging (MRI) is a non-invasive medical tool that physicians use to diagnose patient disease or health conditions. The MRI techniques generally include a powerful magnetic field, radiofrequency pulses, and a computer to produce detailed pictures of all internal body structures. Individual or combined structural MRI biomarkers such as shape and texture of the hippocampus, cortical measurements, and volume measurements are used as biomarkers for the multiclass classification of AD, mild cognitive impairment (MCI). Recently, several machine learning techniques such as SVM (support vector machines), KNN (K-nearest neighbor) algorithm, NN (Neural Network), Ensemble and regression models have been employed to classify AD, MCI. Zubair [3] claimed a method to detect Alzheimer's disease. He made use of a 5 stage ML pipeline process for the detection in which each stage had a sub-stage. Multiple classifiers were applied to this pipeline. He concluded that the random forest Classifier had better performance metrics. Khan et al. [4] made use of the Random-Forest classifier to compare the performance in imputation and non-imputation methods. They observed that the imputation method gives 87% accuracy, and the non-imputation method gives 83% accuracy. It further classified the subjects as demented or non-demented, respectively.

Asim et al. [5] proposed a multi-atlas approach to detect Alzheimer's disease by utilizing the unique features extricated from every atlas template and the joined characteristics of the two atlases using PCA and casted-off SVM for classification. They achieved 94% accuracy for AD vs. CN, 76.5% accuracy for CN vs. MCI, and 75.5% accuracy for MCI vs. AD. They observed that the multi-atlas approach showed

better results than the single-atlas approach. Alam [6] stated in his paper that early-stage detection can prevent the spread of disease. He made use of structural magnetic resonance (MRI) for capturing brain images from the database. He postulated the use of kernels for projecting the data onto the available linear space. He then applied a Support Vector Machine (SVM) for classifying the data. He obtained a good accuracy of 93.85% for his classification with high sensitivity and specificity.

Moein [7] first applied voxel morphometry analysis for capturing some of the most crucial MRI images. He then carried out a principal component analysis on the features extracted. Then a hybrid manifest learning framework was presented in the given subspace. Then, a label propagation model was applied to classify into two types as normal and mild by taking a chunk of training data. The model provided a whooping accuracy of 93.86% for the given classification. Kumar Lama [8] made a collaborative approach to distinguish AD from different other diseases. He made use of SMR's to determine AD amongst mild cognitive impairment and Health Control. He used three algorithms, namely RELM, SVM and IVM, for this segregation process. In addition to that, a discriminative approach including kernels is provided to tackle various data distributions which are complex. He concluded from his classification approach that RELM had better performance metrics.

Dr. Bryan [9] saw that varieties among intersite and multi-vendor estimations restricted AI applications for cerebral bloodstream imaging strategies in Alzheimer's Disease. Such types can be vigorously standardized in human vision but need significant advances in figuring out how to dodge perils from overlooked and underrepresented measurable blunders. Escudero et al. [10] proposed a ML approach using biomarkers in their paper. They tested a personalized classifier for the disease using a method learning locally weighted and biomarkers. The methodology attempts to classify the subject first and then later decides which biomarker to order. They classified the patients who were MCI who advanced to AD inside a year against the individuals who didn't.

A given 3D MR image is taken and is visualized in three orthogonal directions i.e., Axial, Coronal and Sagittal directions. The grey matter and white matter of the brain are separated from the 3D brain image and single slice extraction is performed. Skull stripping [11] is performed on these 2D slices as a pre-processing step to remove non-

cerebral tissues like skull, scalp, and dura from brain images. As part of the feature extraction, first-order statistical features are extracted from the 2D slices, for both white matter and grey matter slices separately. The correlation matrix heatmap of all of the features is prepared to represent the interdependence between them. The principal component analysis is applied to these features as part of the feature reduction step to select the most prominent features. Pre-processing of the data is then performed. Four different classifiers are chosen here to classify the presence of Alzheimer's disease based on the prominent features selected, namely, Logistic Regression, SVM, Naive Bayes and Adaboost classifier on both grey matter and white matter data individually in axial, coronal and sagittal directions. Comparisons between the efficiencies of these classifiers are studied and analyzed.

2.2 Limitations of Existing work

Deep learning, a state-of-the-art machine learning approach, has shown outstanding performance over traditional machine learning in identifying intricate structures in complex high-dimensional data, especially in the domain of computer vision. The application of deep learning to early detection and automated classification of Alzheimer's disease (AD) has recently gained considerable attention, as rapid progress in neuroimaging techniques has generated large-scale multimodal neuroimaging data. A systematic review of publications using deep learning approaches and neuroimaging data for diagnostic classification of AD was performed. A PubMed and Google Scholar search was used to identify deep learning papers on AD published between January 2013 and July 2018. These papers were reviewed, evaluated, and classified by algorithm and neuroimaging type, and the findings were summarized. Of 16 studies meeting full inclusion criteria, 4 used a combination of deep learning and traditional machine learning approaches, and 12 used only deep learning approaches. The combination of traditional machine learning for classification and stacked auto-encoder (SAE) for feature selection produced accuracies of up to 98.8% for AD classification and 83.7% for prediction of conversion from mild cognitive impairment (MCI), a prodromal stage of AD, to AD. It remains a major challenge to select best features that represent characteristics useful to discern between AD, MCI and NC. In recent years, a content-based image retrieval system and medical image classification were vigorously interactive to each other and were used for detecting AD or MCI. Sometimes, the

classification by one specific classifier does not provide the desired result due to certain factors, so there were attempts to use an ensemble of classifiers and a multistage classifier for more correct classification results. The ensemble approach uses a voting technique principally, and the multistage classifier uses output to one stage classifier as input to the next stage classifier. In this way, the advantages of individual classifiers can be combined and disadvantages can be overcome.

Deep learning approaches, such as convolutional neural network (CNN) or recurrent neural network (RNN), that use neuroimaging data without pre-processing for feature selection have yielded accuracies of up to 96.0% for AD classification and 84.2% for MCI conversion prediction. The best classification performance was obtained when multimodal neuroimaging and fluid biomarkers were combined. Deep learning approaches continue to improve in performance and appear to hold promise for diagnostic classification of AD using multimodal neuroimaging data. AD research that uses deep learning is still evolving, improving performance by incorporating additional hybrid data types, such as—omics data, increasing transparency with explainable approaches that add knowledge of specific disease-related features and mechanisms.

CHAPTER-3

SOFTWARE AND HARDWARE SPECIFICATIONS

Functional requirements

The functional requirements for a system describe what the system should do. Those requirements depend on the type of software being developed, the expected users of the software. These are statement of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situation.

- Become familiar with dataset characteristics and data
- Convert the dataset(images) into pixels and send it as input data for our model.
- Check to see if the input data is resulting to Alzheimer's or not.
- If it is resulting to Alzheimer’s, let the patient know that he is suffering from Alzheimer's
- Provide the percentage accuracy of the model

Non-Functional requirements

Non-functional requirements are requirements that are not directly concerned with the specified function delivered by the system. They may relate to emergent system properties such as reliability, response time and store occupancy. Some of the non-functional requirements related with this system are hereby below:

- The system should be implemented in python.
- The system should produce graphs showing performance.
- The model should be able to work easily on huge amounts of data and predict effectively.
- The model should be able to perform well on all platforms.

3.1 Software Requirements

Following are the software requirements necessary of the project:

- OS: Windows/Mac/Linux
- Platforms: Anaconda/Jupyter Notebook,Google Colab
- Software: Python 3.7
- Packages: pandas, seaborn, matplotlib, NumPy, TensorFlow, keras

3.2 Hardware Requirements

Following are the hardware requirements that is most important for the project:

- RAM: 4GB
- Processor: Intel I5 octa-core (1.5GHz - 2.6GHz)
- Secondary Storage: 256GB
- Graphics card: 1650 or 1660TI

CHAPTER-4

PROPOSED SYSTEM DESIGN

4.1 Proposed Method

Deep Learning is known to be learning a hierarchical set of representations such that it learns low mid and high-level features. Deep neural networks can adapt to more complex data sets. It's better in generalizing previously unseen data because of its multiple layers. Different algorithms use Deep Learning's fundamental expertise and use diverse datasets to train and test these algorithms. Like neurons in humans, deep learning has layers that help the model or algorithm learn and process the data. These layers process the data given to them as the input and learn by processing the input while traveling through the layers. When it passes out the last layer, an activation function is applied at last, and finally, we get the predicted output from the deep learning model. This gives us the training accuracy, and then when we take another similar type of dataset, we can predict or detect from the learned deep learning model whatsoever we want. Well, this is what deep learning does in simple working terms. The advantages of training a deep learning model from scratch and of transfer learning are subjective. It depends a lot on the problem you are trying to solve, the time constraints, the availability of data and the computational resources you have. You can use a pretrained model (for example, DenseNet169 or VGG19) as the backbone for obtaining image features and train a classifier (for example a two layered neural network) on top of it. Here, you keep the backbone part obtained from the pretrained model fixed and only allow the parameters of the classifier to change. This approach is ideal when you want to train a model quickly or without much computational resources. The performance of in this case might not be the best because the pretrained backbone may suffer from domain adaptation. You can train a deep learning model (for example DenseNET169 or VGG19) from scratch for your problem. This means that you initialize the model with new parameters i.e., not obtained from a pretrained model. This requires more computational resources (or time) to train but learns all the parameters from the training data. If the training data is sufficiently enough, a model

equivalent to the model in approach 1 when trained using this approach should perform better.

4.1.1 Dataset

The data is taken from an open online dataset library known as Kaggle, and the dataset hasn't been used by various other research projects and studies yet. It is an open-source dataset. This dataset contains almost 6,000 images distributed over two classes labelled as

1. Demented
2. Non-Demented

The features are then distributed into 80% train dataset and 20% test dataset. 80% of the data is used in training means that each deep learning model has two phases that are training and testing, where it predicts the data that is provided to it. Both the models use the same dataset separated from the original Kaggle dataset and are divided in 8:2 ratio, which has 80% training and 20% validation dataset. The datasets have to have the same kind of distribution so that there is no such kind of discrepancy in the comparison of the prediction that both the models had a different type of input. This removes the question upon both the models and brings them to the same inspection level that both were using, not the same dataset, and was trained and tested upon the exact distribution of the dataset. The main inspiration behind this Dataset is to make a very highly accurate model predict the stage of Alzheimer’s.

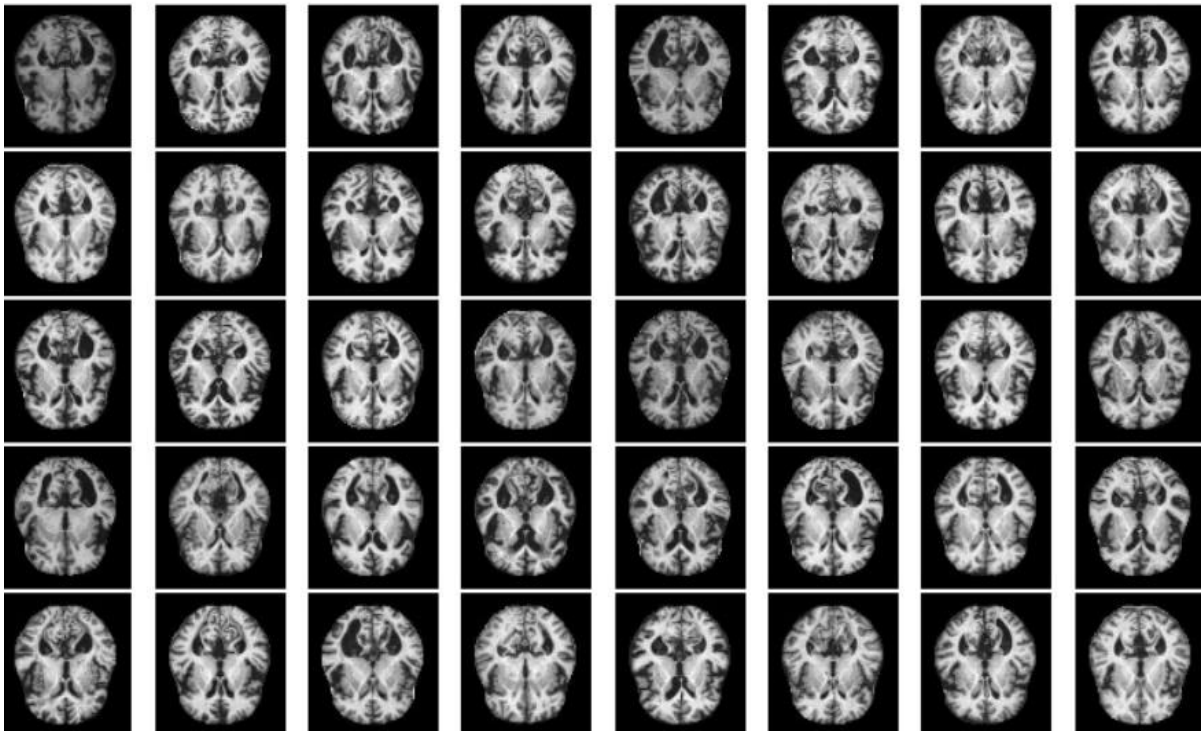

Fig-4.1: Demented Alzheimer’s Scan.

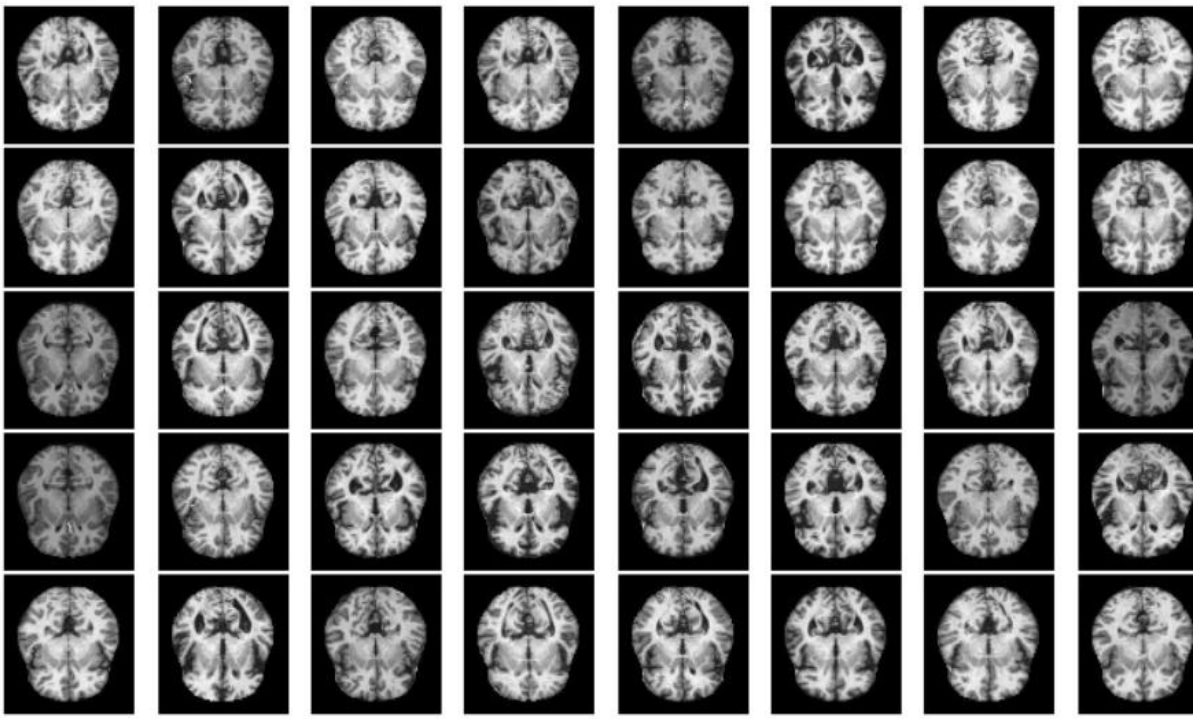

Fig-4.2: Non-Demented Alzheimer’s Scan.

4.1.2 Proposed Methodology

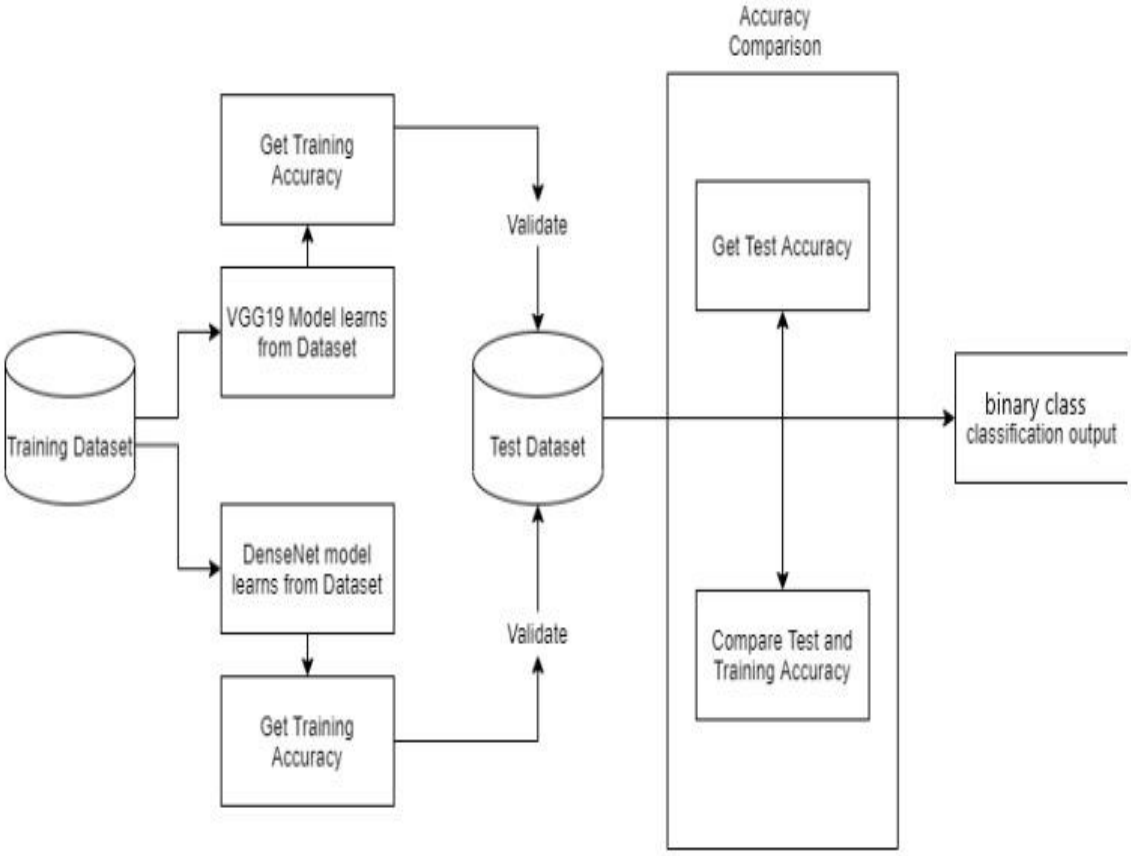

Fig-4.3: Proposed Methodology.

The proposed methodology diagram shows us a conceptual and behavioral view of the system. It is just a view that shows us how the database is used in taking the dataset and then how this data is used up in our project modules to train the different models. In the diagram above, we can see the data is being taken from the training dataset and then provided to the models. Then it is validated against the test dataset to get the testing or validation accuracy.

After the accuracy is compared, the diseased images are taken from the dataset, The classification done is of two types namely Demented and Non-Demented. Also, the architecture diagram shows us the various modules working together in the project and how they are integrated to provide us the desired output, and how all the modules need to be interconnected to make the project work in unison.

4.1.3 VGG19

A. Zisserman and K. Simonyan of the University of Oxford proposed VGG19. It is one of the convolutional neural system models, consisting of 16 convolutional layers along with three fully connected layers. It's shown to yield excellent results even though there are a large number of groups to classify. According to A. Zisserman and K. Simonyan's analysis, the model was able to classify a dataset of about 1000 classes with an accuracy of 92.7 percent. VGG19 is a well-known classifying model that can classify a wide range of classes and is used in a variety of medical research projects. It has a high level of accuracy in predicting things like vehicles and trees. VGG19 is now being used on a wider scale in medical datasets to predict smaller groups, such as breast cancer detection, macular edema detection, brain tumor detection, and so on. This is one of the reasons that the same model has been used for the classification in this study. It also offers a standard method of constructing a classifier, which is helpful in most studies since it employs simple Convolutional and Max Pooling layers in its model construction.

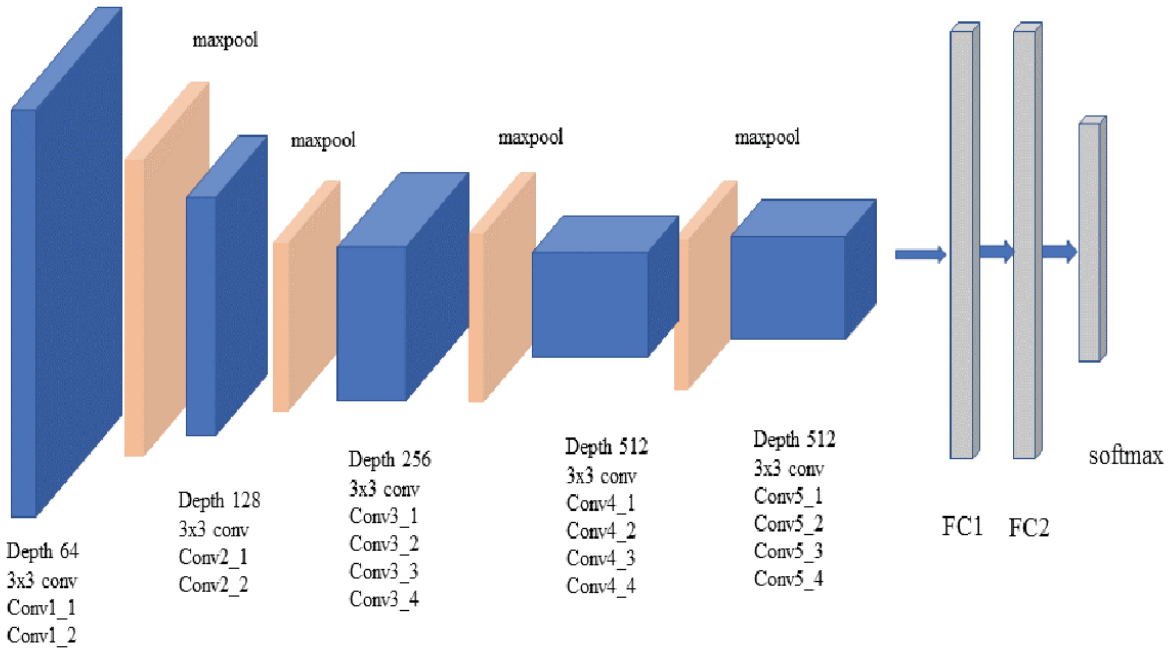

Fig-4.4: VGG19 Architecture.

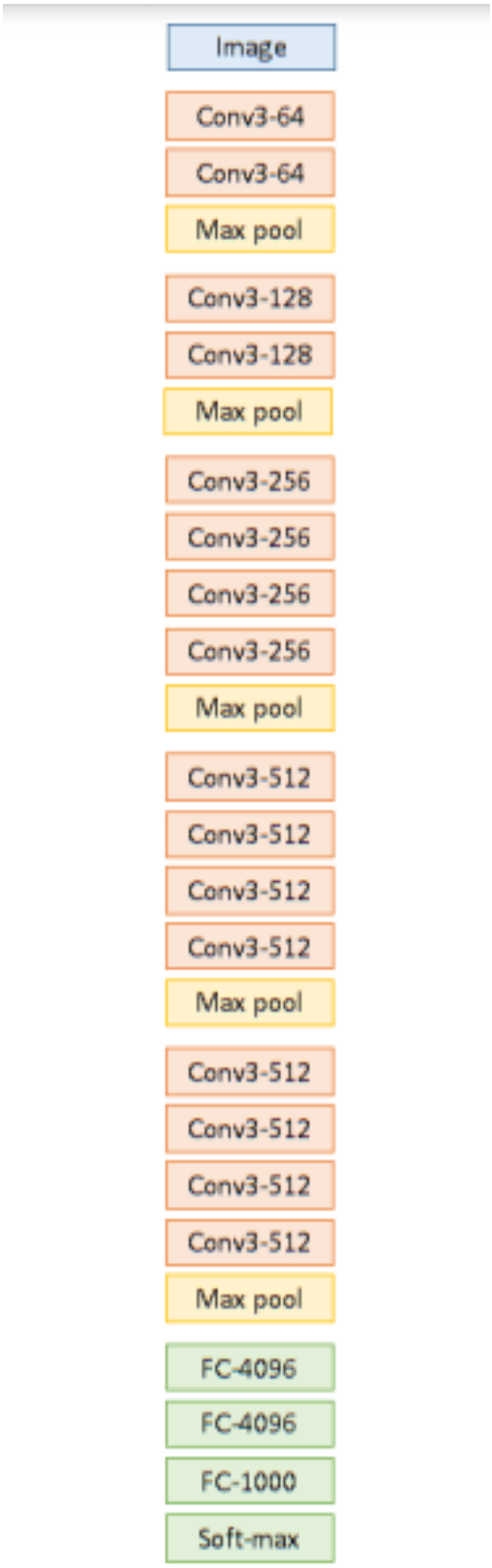

Fig-4.5: Layers in VGG19 Architecture.

VGG19 is a variant of VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer).

Convolution layer:

Convolutional layers in a convolutional neural network systematically apply learned filters to input images in order to create feature maps that summarize the presence of those features in the input. Convolutional layers prove very effective, and stacking convolutional layers in deep models allows layers close to the input to learn low-level features (e.g., lines) and layers deeper in the model to learn high-order or more abstract features, like shapes or specific objects.

MaxPool layer:

The addition of a pooling layer after the convolutional layer is a common pattern used for ordering layers within a convolutional neural network that may be repeated one or more times in a given model. The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps. Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, it is almost always 2×2 pixels applied with a stride of 2 pixels. This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g., each dimension is halved, reducing the number of pixels or values in each feature map to one quarter the size. Two common functions used in the pooling operation are:

Average Pooling: Calculate the average value for each patch on the feature map.

Maximum Pooling (or Max Pooling): Calculate the maximum value for each patch of the feature map.

Fully connected layer:

Fully Connected Layer is simply, feed forward neural networks. Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.

Architecture:

- A fixed size of (229 * 229) RGB image was given as input to this network which means that the matrix was of shape (229,229,3).
- The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set.
- Used kernels of (3 * 3) size with a stride size of 1 pixel, this enabled them to cover the whole notion of the image.
- Spatial padding was used to preserve the spatial resolution of the image.
- Max pooling was performed over a 2 * 2-pixel windows with sride 2.
- This was followed by Rectified linear unit (ReLu) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used tanh or sigmoid functions this proved much better than those.
- Implemented three fully connected layers from which first two were of size 4096 and after that a layer with 1000 channels for 1000-way ILSVRC classification and the final layer is a softmax function.

4.1.4 DenseNET169

Convolutional neural networks are popular for image processing because of a number of technical advantages compared to fully connected neural networks or other architectures. CNNs are composed of convolution layers followed by subsampling layers. The convolution layer allows detecting features (or patterns) in the image, such as edges. In addition, a subsampling layer (such as avg pooling, max pooling, down-sampling, up-sampling) allows the resolution to be reduced at the position of the patterns. In the end, we are not concerned with the exact location of the patterns in the image but with their relationship to other patterns [12].

In a standard Convolutional neural network, the input image is passed from layers of the network to obtain an output of a predicted label in which the forward pass is quite straightforward.

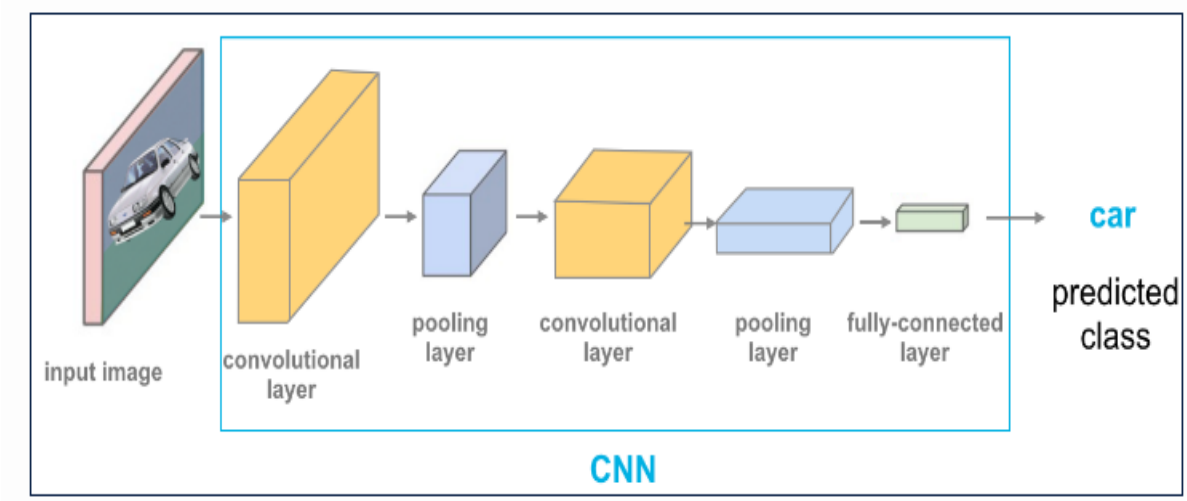

Fig-4.6: CNN

Each convolutional layer except the first one (which takes in the input image), takes in the output of the previous convolutional layer and produces an output feature map that is then passed to next convolutional layer. For L layers, there are L direct connections one between each layer and its subsequent layer.

For image classification, new CNN architectures have been created and tested using databases such as ImageNet. These include VGG, ResNet, AlexNet, DenseNet. Among the models available in the Keras API (Keras Applications) such as VGG, ResNet, DenseNet, InceptionV3 and MobileNet. DenseNet-169 was chosen because despite having a depth of 169 layers it is relatively low in parameters compared to other models, and the architecture handles the vanish gradient problem well.

Every convolutional layer, except the first layer gets the output through the trailing convolutional layer and produces a feature map output which is forwarded to the leading convolutional layer. The DenseNet is the modification of the standard CNN model. We evaluated the use of DenseNet-169 pre-trained with ImageNet weights and modified the last layer as will be specified later. DenseNet is composed of Dense blocks. In those blocks, the layers are densely connected together: Each layer receive in input all previous layer's output feature maps. A dense block is a group of layers connected to all their previous layers.

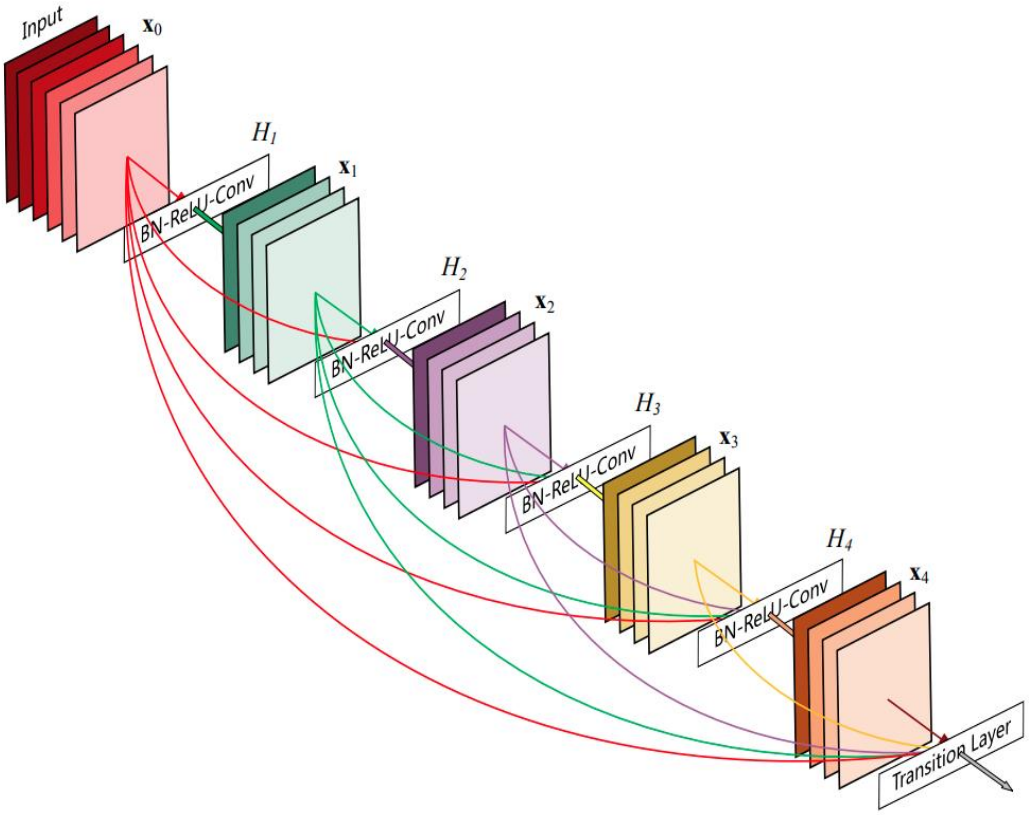

Fig-4.7: DenseNet169 Architecture.

In a DenseNet architecture, each layer is connected to every other layer, hence the name Densely Connected Convolutional Network. For L Layers, there can be L(L+1)/2 connections. For every layer, all the trailing layer feature maps can be used as an input, and its own mapping feature can be pushed as an input for the leading layers. DenseNets essentially connect every layer to every other layer. This is the main idea that is extremely powerful. The input of a layer inside DenseNet is the concatenation of feature maps from previous layers.

To facilitate both down-sampling in the architecture and feature concatenation the network is divided into multiple densely connected dense blocks. Inside the dense blocks, the feature map size remains the same. The input of shape *224x224x3* is down sampled to *7x7x512* towards the end of the network.

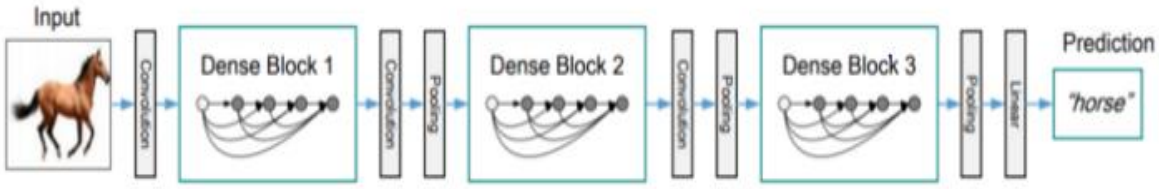

Fig-4.8: A DenseNet Architecture with 3 dense blocks.

Dividing the network into densely connected blocks solves the problem. Now, the Convolution and pooling operations outside the dense blocks can perform the down sampling operation and inside the dense block we can make sure that the size of the feature maps is the same to be able to perform feature concatenation.

The layers between the dense blocks are called as transition layers which do the convolution and pooling. The transition layers used in the DenseNet architecture consist of a batch-norm layer, 1x1 convolution followed by a 2x2 average pooling layer. Instead of summing the residual like in ResNet, DenseNet concatenates all the feature maps. It would be impracticable to concatenate feature maps of different sizes (although some resizing may work). Thus, in each dense block, the feature maps of each layer have the same size. However down-sampling is essential to CNN. Transition layers between two dense blocks assure this role. A transition layer is made up of

1. Batch Normalization
2. 1x1 Convolution
3. Average pooling

Some of the advantages of DenseNet are

- The vanishing gradient problem is alleviated by densenet.
- Feature propagation is strengthened
- Reusability of feature
- Number of parameters is reduced

4.2 Class Diagram

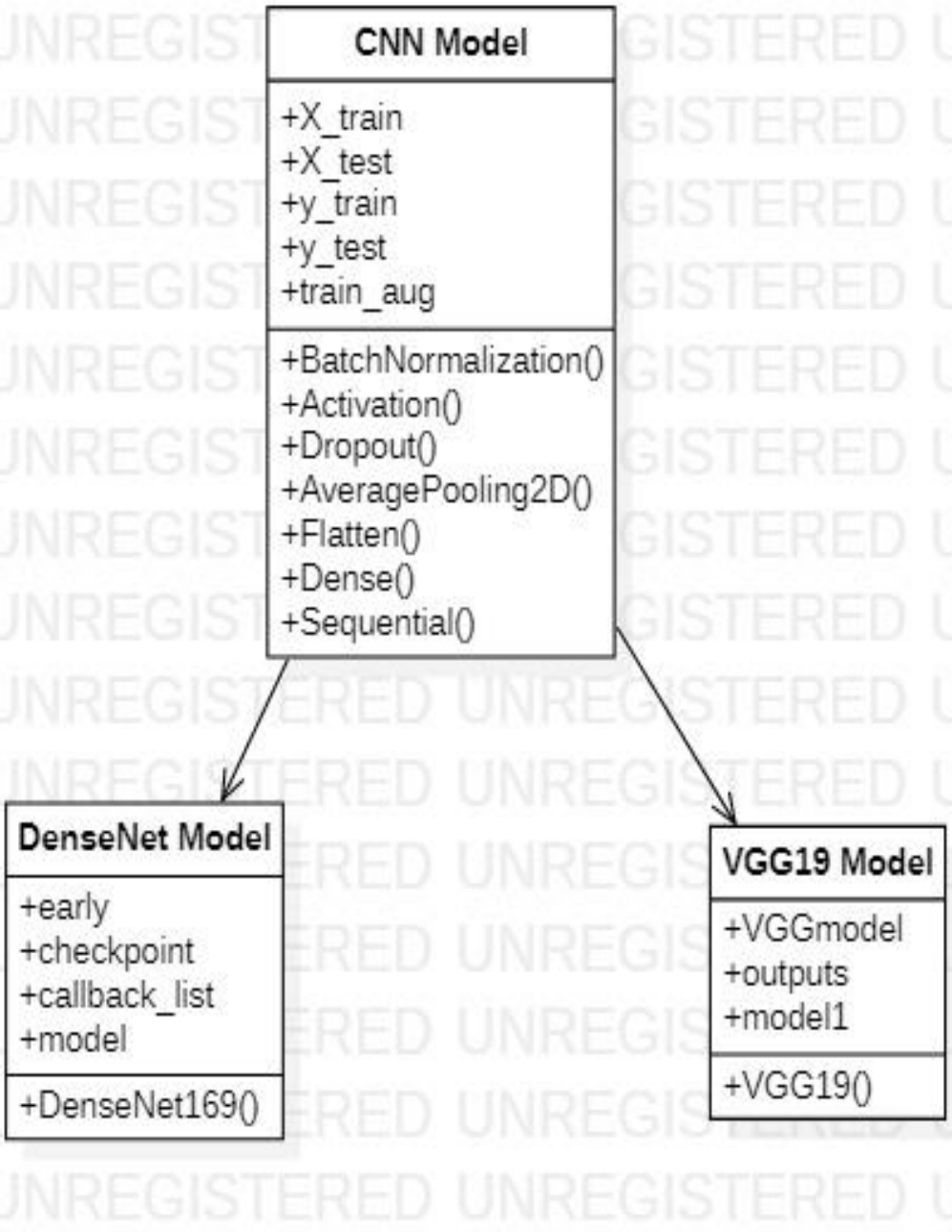

Fig-4.9: Class Diagram.

This class diagram consists of three modules CNN model, DenseNet model and VGG19 model. All the classes contain appropriate attributes and operations used in that class. DenseNet model and CNN model does the main task in predicting Alzheimer's disease using MRI images.

4.3 Use Case Diagram

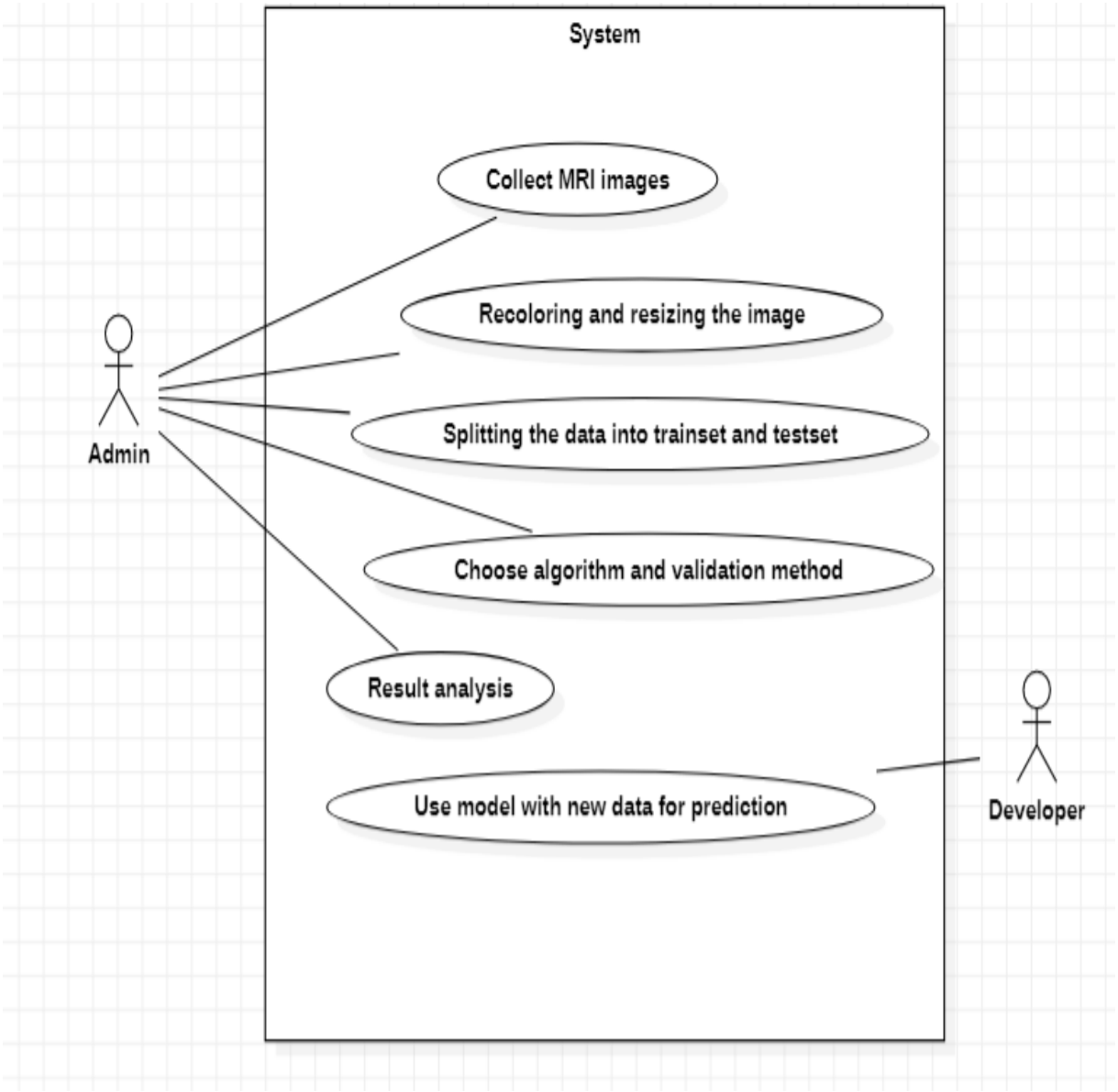

Fig-4.10: Use Case Diagram.

This diagram shows the actor (admin) who interacts and performs various operations on the system. The operations include collecting MRI images, recoloring and resizing the images, splitting the data into train set and test set, choosing appropriate algorithm and analyzing the results.

4.4 Activity Diagram

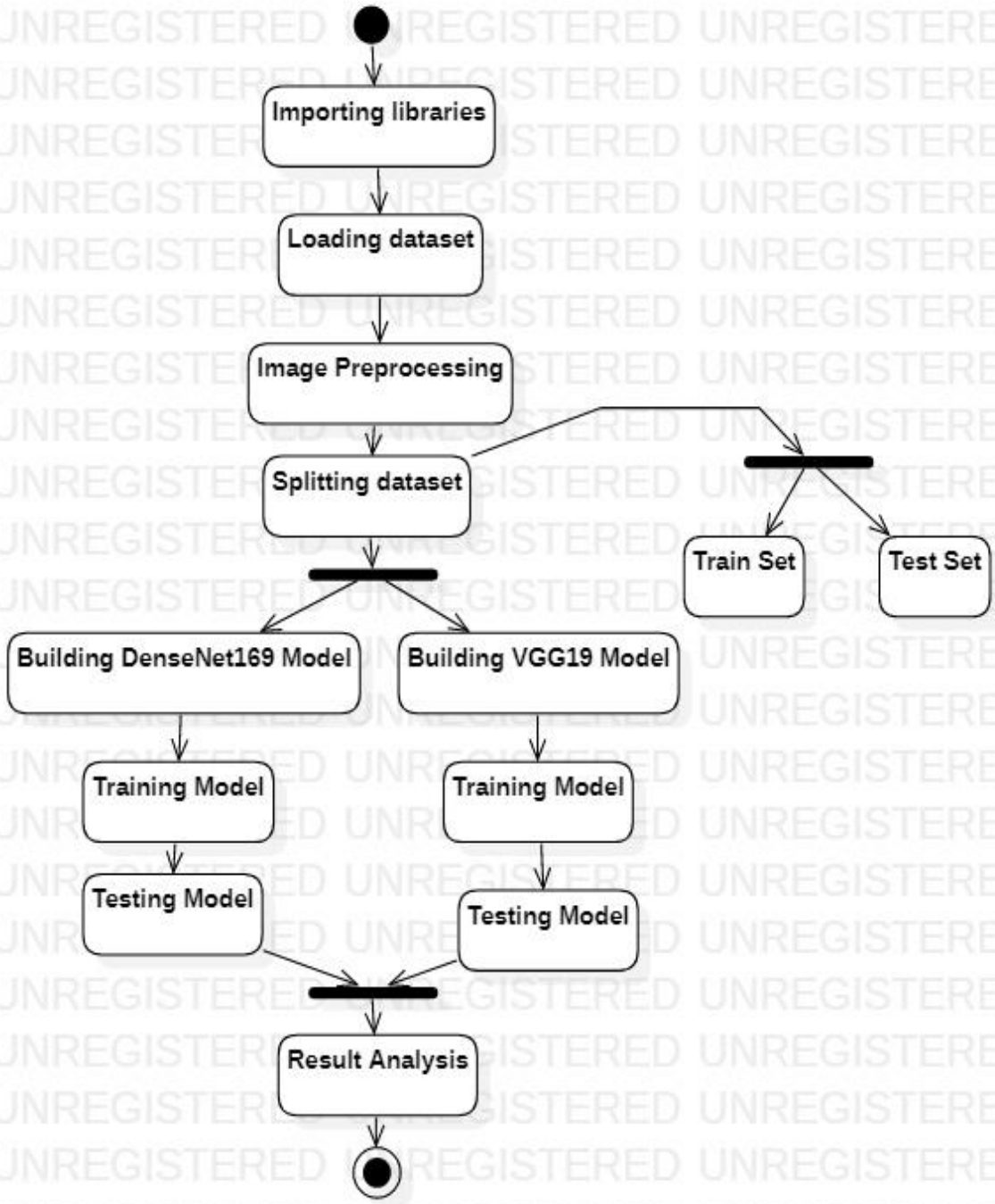

Fig-4.11: Activity Diagram.

This diagram describes the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. This diagram shows the flow of operations in the system and demonstrates it clearly with a flowchart.

4.5 Sequence Diagram

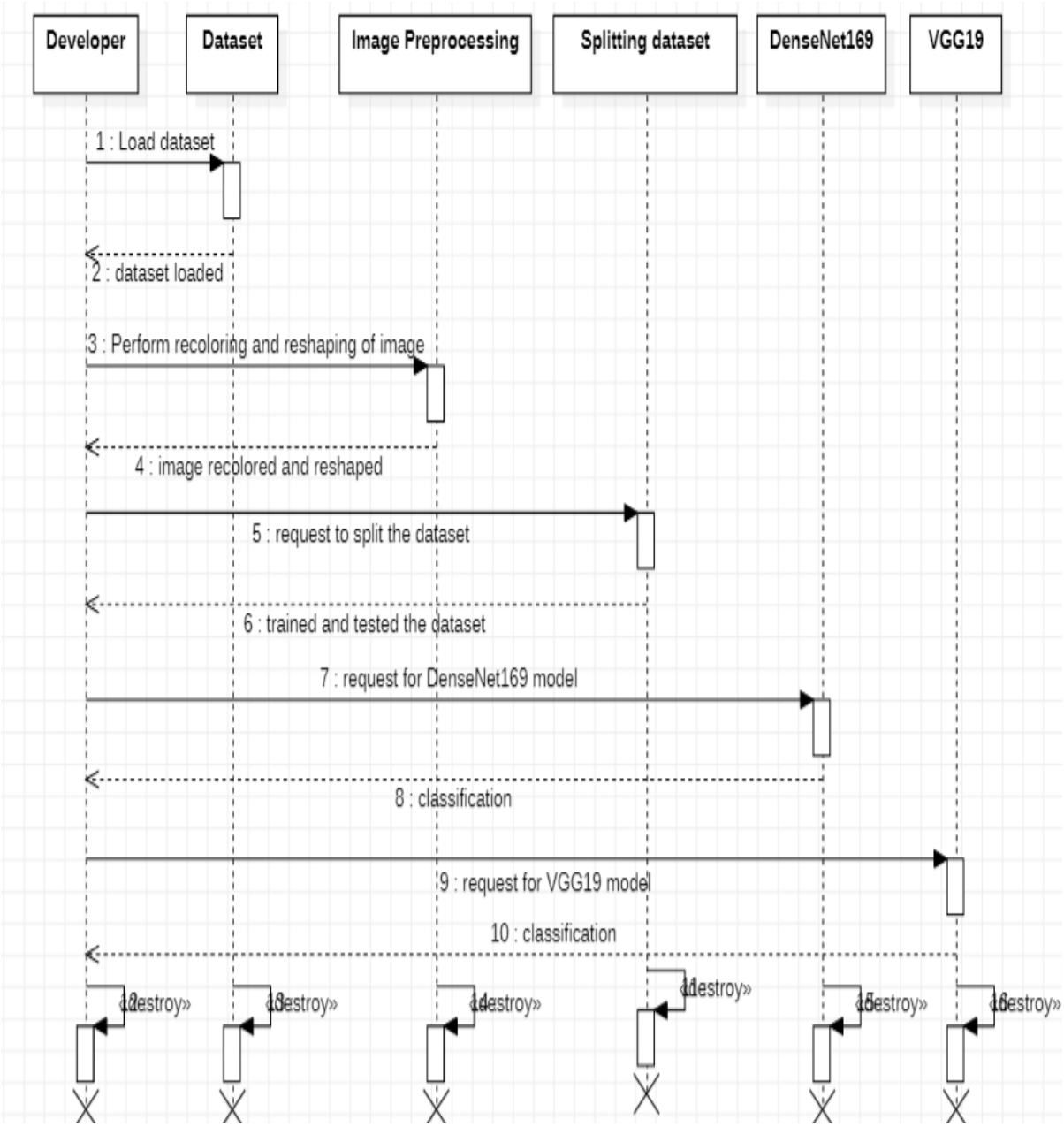

Fig-4.12: Sequence Diagram.

4.6 System Architecture

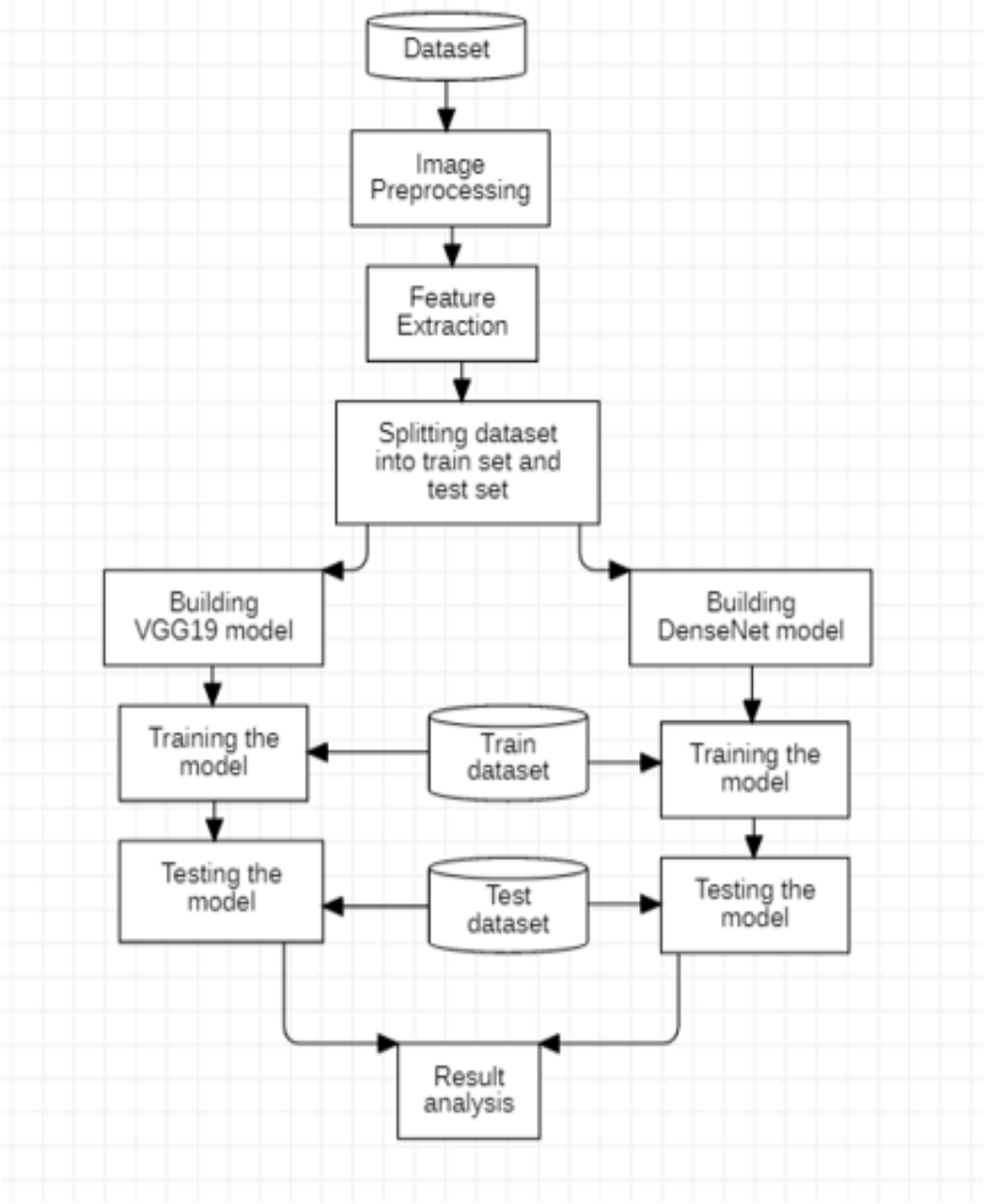

Fig-4.13: System Architecture.

The architecture diagram depicts the system architecture and give the major components in the system and how they are related. The major components in image preprocessing are image recoloring and image resizing. Each CNN model used in proposed model also have different architectures which also have an impact on performance of the model.

4.7 Technology Description

4.7.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics and simple, easy to learn syntax that 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.

Many major libraries and API-powered services have Python bindings or wrappers, letting Python interface freely with those services or directly use those libraries. Even though scripting and automation cover a large chunk of Python's use cases (more on that later), Python is also used to build professional-quality software, both as standalone applications and as web services. Python may not be the fastest language, but what it lacks in speed, it makes up for in versatility.

4.7.2 Jupyter Notebook

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

A Jupyter Notebook can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) through "Download As" in the web interface, via the nbconvert library or "jupyter nbconvert" command line interface in a shell. To simplify visualisation of Jupyter notebook documents on the web, the nbconvert library is provided as a service through

NbViewer which can take a URL to any publicly available notebook document, convert it to HTML on the fly and display it to the user.

4.7.3 NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Num array.

The core functionality of NumPy is its "ndarray", for n-dimensional array, data structure. These arrays are strided views on memory. In contrast to Python's built-in list data structure, these arrays are homogeneously typed: all elements of a single array must be of the same type. Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries.

4.7.4 Pandas

Pandas is an open-source library that is made mainly for working with relational or labelled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users. Pandas is a Python library used for working with data sets.It has functions for analyzing, cleaning, exploring, and manipulating data.

Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science. Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

4.7.5 Matplotlib

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pyl ab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. Pyplot is a Matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

4.7.6 Seaborn

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas.

Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs, so that we can switch between different visual representations for same variables for better understanding of dataset.

Plots are basically used for visualizing the relationship between variables. Those variables can be either be completely numerical or a category like a group, class or division. Seaborn divides plot into different categories such as relational plots, categorical plots, distribution plots, regression plots, matrix plots, multi-plot grids etc.

4.7.7 Plotly

The Plotly Python library is an interactive open-source library. This can be a very helpful tool for data visualization and understanding the data simply and easily. plotly graph objects are a high-level interface to plotly which are easy to use. It can plot various types of graphs and charts like scatter plots, line charts, bar charts, box plots, histograms, pie charts, etc. Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in a large number of data points. It is visually attractive that can be accepted by a wide range of audiences. It allows us for the endless customization of our graphs that makes our plot more meaningful and understandable for others.

4.7.8 sklearn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

4.7.9 Tensorflow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow was developed by the Google Brain team for internal Google use in research and production.

TensorFlow can be used in a wide variety of programming languages, most notably Python, as well as JavaScript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors. TensorFlow serves as the core platform and library for machine learning. TensorFlow's APIs use Keras to allow users to make their own machine learning models. In addition to building and training their model, TensorFlow can also help load the data to train the model, and deploy it using TensorFlow Serving.

4.7.10 Keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

4.7.11 OpenCV

OpenCV is a library of programming functions primarily geared toward real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source Apache 2 License. Starting with 2011, OpenCV features GPU acceleration for real-time operations.

CHAPTER-5

IMPLEMENTATION AND TESTING

5.1 Code Snippets

5.1.1 Image Preprocessing

```
Dem_labels = []
NonDem_labels = []

Dem_images=[]
NonDem_images=[]

for i in range(len(Demfiles)):
  image = cv2.imread(Demfiles[i])
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  image = cv2.resize(image,(229,229))
  Dem_images.append(image)
  Dem_labels.append('Demented')
for i in range(len(NonDemfiles)):
  image = cv2.imread(NonDemfiles[i])
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  image = cv2.resize(image,(229,229))
  NonDem_images.append(image)
  NonDem_labels.append('NonDemented')
```

Fig-5.1: Image Preprocessing.

5.1.2 Splitting dataset into trainset and testset

```
Dem_x_train, Dem_x_test, Dem_y_train, Dem_y_test = train_test_split(
    Dem_images, Dem_labels, test_size=0.2)
NonDem_x_train, NonDem_x_test, NonDem_y_train, NonDem_y_test = train_test_split(
    NonDem_images, NonDem_labels, test_size=0.2)


X_train = np.concatenate((NonDem_x_train, Dem_x_train), axis=0)
X_test = np.concatenate((NonDem_x_test, Dem_x_test), axis=0)
y_train = np.concatenate((NonDem_y_train, Dem_y_train), axis=0)
y_test = np.concatenate((NonDem_y_test, Dem_y_test), axis=0)

y_train = LabelBinarizer().fit_transform(y_train)
y_train = to_categorical(y_train)

y_test = LabelBinarizer().fit_transform(y_test)
y_test = to_categorical(y_test)
```

Fig-5.2: Splitting Dataset.

5.1.3 Model Building

DenseNet169 Model

In [11]:
```
base_model = DenseNet169(input_shape=(229,229,3),
                         include_top=False,
                         weights="imagenet")
```

In [12]:
```
for layer in base_model.layers:
    layer.trainable=False
```

In [13]:
```
# Building Model

model=Sequential()
model.add(base_model)
model.add(Dropout(0.5))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(2048,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1024,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(2,activation='softmax'))
```

In [14]:
```
model.summary()
```

In [15]:
```
model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
)
```

In [16]:
```
train_aug = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)
```

Fig-5.3: Building DenseNet169

VGG19 Model

```
VGGmodel = VGG19(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(229, 229, 3)))

outputs = VGGmodel.output
outputs = Flatten(name="flatten")(outputs)
outputs = Dropout(0.4)(outputs)
outputs = Dense(2, activation="softmax")(outputs)

model1 = Model(inputs=VGGmodel.input, outputs=outputs)

for layer in VGGmodel.layers:
    layer.trainable = False

model1.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
)
```

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
filepath = 'Weights_AD_Model_VGG19.hdf5'

earlystopping = EarlyStopping(monitor = 'accuracy',
                              mode = 'max' ,
                              patience = 15,
                              verbose = 1)

chckpoint    = ModelCheckpoint(filepath,
                               monitor = 'accuracy',
                               mode='max',
                               save_best_only=True,
                               verbose = 1)


callback_list2 = [earlystopping, chckpoint]
```

Fig-5.4: Building VGG19

5.2 Implementation Screenshots

DenseNet169

```
history = model.fit(train_aug.flow(X_train, y_train, batch_size=32),
                    validation_data=(X_test, y_test),
                    validation_steps=len(X_test) / 64,
                    steps_per_epoch=len(X_train) / 64,
                    epochs=100,
                   callbacks = [early,checkpoint])
```

```
Epoch 1/100
13/12 [==============================] - ETA: -3s - loss: 0.7922 - accuracy: 0.7356
Epoch 00001: accuracy improved from -inf to 0.73558, saving model to ./best_weights.hdf5
13/12 [==============================] - 169s 13s/step - loss: 0.7922 - accuracy: 0.7356 - val_loss: 1.3786 - val_accuracy: 0.8750
Epoch 2/100
13/12 [==============================] - ETA: -2s - loss: 0.6422 - accuracy: 0.7716
Epoch 00002: accuracy improved from 0.73558 to 0.77163, saving model to ./best_weights.hdf5
13/12 [==============================] - 126s 10s/step - loss: 0.6422 - accuracy: 0.7716 - val_loss: 1.1302 - val_accuracy: 0.8800
Epoch 3/100
13/12 [==============================] - ETA: -2s - loss: 0.4207 - accuracy: 0.8486
Epoch 00003: accuracy improved from 0.77163 to 0.84856, saving model to ./best_weights.hdf5
13/12 [==============================] - 128s 10s/step - loss: 0.4207 - accuracy: 0.8486 - val_loss: 0.7351 - val_accuracy: 0.8750
Epoch 4/100
13/12 [==============================] - ETA: -2s - loss: 0.4738 - accuracy: 0.8317
Epoch 00004: accuracy did not improve from 0.84856
13/12 [==============================] - 104s 8s/step - loss: 0.4738 - accuracy: 0.8317 - val_loss: 0.5956 - val_accuracy: 0.8700
Epoch 5/100
13/12 [==============================] - ETA: -2s - loss: 0.4587 - accuracy: 0.8269
Epoch 00005: accuracy did not improve from 0.84856
13/12 [==============================] - 108s 8s/step - loss: 0.4587 - accuracy: 0.8269 - val_loss: 0.3623 - val_accuracy: 0.8800
Epoch 6/100
13/12 [==============================] - ETA: -2s - loss: 0.3892 - accuracy: 0.8221
Epoch 00006: accuracy did not improve from 0.84856
13/12 [==============================] - 106s 8s/step - loss: 0.3892 - accuracy: 0.8221 - val_loss: 0.3073 - val_accuracy: 0.8700
Epoch 7/100
13/12 [==============================] - ETA: -2s - loss: 0.4482 - accuracy: 0.8462
Epoch 00007: accuracy did not improve from 0.84856
13/12 [==============================] - 107s 8s/step - loss: 0.4482 - accuracy: 0.8462 - val_loss: 0.2877 - val_accuracy: 0.8800
Epoch 8/100
13/12 [==============================] - ETA: -2s - loss: 0.3018 - accuracy: 0.8918
Epoch 00008: accuracy improved from 0.84856 to 0.89183, saving model to ./best_weights.hdf5
13/12 [==============================] - 143s 11s/step - loss: 0.3018 - accuracy: 0.8918 - val_loss: 0.2700 - val_accuracy: 0.8950
Epoch 9/100
13/12 [==============================] - ETA: -2s - loss: 0.3878 - accuracy: 0.8534
Epoch 00009: accuracy did not improve from 0.89183
13/12 [==============================] - 107s 8s/step - loss: 0.3878 - accuracy: 0.8534 - val_loss: 0.2781 - val_accuracy: 0.9050
Epoch 10/100
13/12 [==============================] - ETA: -2s - loss: 0.3406 - accuracy: 0.8798
Epoch 00010: accuracy did not improve from 0.89183
13/12 [==============================] - 105s 8s/step - loss: 0.3406 - accuracy: 0.8798 - val_loss: 0.2986 - val_accuracy: 0.8700
Epoch 11/100
13/12 [==============================] - ETA: -3s - loss: 0.3356 - accuracy: 0.8846
Epoch 00011: accuracy did not improve from 0.89183
13/12 [==============================] - 109s 8s/step - loss: 0.3356 - accuracy: 0.8846 - val_loss: 0.2686 - val_accuracy: 0.8650
Epoch 12/100
13/12 [==============================] - ETA: -2s - loss: 0.4627 - accuracy: 0.8558
Epoch 00012: accuracy did not improve from 0.89183
13/12 [==============================] - 107s 8s/step - loss: 0.4627 - accuracy: 0.8558 - val_loss: 0.2830 - val_accuracy: 0.8700
Epoch 13/100
13/12 [==============================] - ETA: -2s - loss: 0.4337 - accuracy: 0.8438
Epoch 00013: accuracy did not improve from 0.89183
13/12 [==============================] - 110s 8s/step - loss: 0.4337 - accuracy: 0.8438 - val_loss: 0.2212 - val_accuracy: 0.8850
Epoch 14/100
13/12 [==============================] - ETA: -3s - loss: 0.3889 - accuracy: 0.8269
Epoch 00014: accuracy did not improve from 0.89183
13/12 [==============================] - 110s 8s/step - loss: 0.3889 - accuracy: 0.8269 - val_loss: 0.2269 - val_accuracy: 0.8900
```

```
13/12 [==============================] - ETA: -2s - loss: 0.3337 - accuracy: 0.8630
Epoch 00015: accuracy did not improve from 0.89183
13/12 [==============================] - 106s 8s/step - loss: 0.3337 - accuracy: 0.8630 - val_loss: 0.2150 - val_accuracy: 0.8600
Epoch 16/100
13/12 [==============================] - ETA: -2s - loss: 0.2887 - accuracy: 0.8870
Epoch 00016: accuracy did not improve from 0.89183
13/12 [==============================] - 118s 9s/step - loss: 0.2887 - accuracy: 0.8870 - val_loss: 0.2291 - val_accuracy: 0.8400
Epoch 17/100
13/12 [==============================] - ETA: -2s - loss: 0.3332 - accuracy: 0.8702
Epoch 00017: accuracy did not improve from 0.89183
13/12 [==============================] - 102s 8s/step - loss: 0.3332 - accuracy: 0.8702 - val_loss: 0.2141 - val_accuracy: 0.8550
Epoch 18/100
13/12 [==============================] - ETA: -2s - loss: 0.3080 - accuracy: 0.8582
Epoch 00018: accuracy did not improve from 0.89183
13/12 [==============================] - 102s 8s/step - loss: 0.3080 - accuracy: 0.8582 - val_loss: 0.2096 - val_accuracy: 0.9200
Epoch 19/100
13/12 [==============================] - ETA: -2s - loss: 0.3404 - accuracy: 0.8534
Epoch 00019: accuracy did not improve from 0.89183
13/12 [==============================] - 104s 8s/step - loss: 0.3404 - accuracy: 0.8534 - val_loss: 0.2064 - val_accuracy: 0.8950
Epoch 20/100
13/12 [==============================] - ETA: -3s - loss: 0.3230 - accuracy: 0.8870
Epoch 00020: accuracy did not improve from 0.89183
13/12 [==============================] - 109s 8s/step - loss: 0.3230 - accuracy: 0.8870 - val_loss: 0.1966 - val_accuracy: 0.9000
Epoch 21/100
13/12 [==============================] - ETA: -2s - loss: 0.2974 - accuracy: 0.8774
Epoch 00021: accuracy did not improve from 0.89183
13/12 [==============================] - 104s 8s/step - loss: 0.2974 - accuracy: 0.8774 - val_loss: 0.1947 - val_accuracy: 0.8950
Epoch 22/100
13/12 [==============================] - ETA: -2s - loss: 0.2460 - accuracy: 0.8966
Epoch 00022: accuracy improved from 0.89183 to 0.89663, saving model to ./best_weights.hdf5
13/12 [==============================] - 145s 11s/step - loss: 0.2460 - accuracy: 0.8966 - val_loss: 0.1869 - val_accuracy: 0.8950
Epoch 23/100
13/12 [==============================] - ETA: -2s - loss: 0.2429 - accuracy: 0.9038
Epoch 00023: accuracy improved from 0.89663 to 0.90385, saving model to ./best_weights.hdf5
13/12 [==============================] - 131s 10s/step - loss: 0.2429 - accuracy: 0.9038 - val_loss: 0.1841 - val_accuracy: 0.8900
Epoch 24/100
13/12 [==============================] - ETA: -2s - loss: 0.2586 - accuracy: 0.9062
Epoch 00024: accuracy improved from 0.90385 to 0.90625, saving model to ./best_weights.hdf5
13/12 [==============================] - 127s 10s/step - loss: 0.2586 - accuracy: 0.9062 - val_loss: 0.2043 - val_accuracy: 0.8900
Epoch 25/100
13/12 [==============================] - ETA: -2s - loss: 0.2586 - accuracy: 0.8990
Epoch 00025: accuracy did not improve from 0.90625
13/12 [==============================] - 108s 8s/step - loss: 0.2586 - accuracy: 0.8990 - val_loss: 0.1833 - val_accuracy: 0.8850
Epoch 26/100
13/12 [==============================] - ETA: -3s - loss: 0.2328 - accuracy: 0.9111
Epoch 00026: accuracy improved from 0.90625 to 0.91106, saving model to ./best_weights.hdf5
13/12 [==============================] - 129s 10s/step - loss: 0.2328 - accuracy: 0.9111 - val_loss: 0.1845 - val_accuracy: 0.9000
Epoch 27/100
13/12 [==============================] - ETA: -2s - loss: 0.2733 - accuracy: 0.8918
Epoch 00027: accuracy did not improve from 0.91106
13/12 [==============================] - 117s 9s/step - loss: 0.2733 - accuracy: 0.8918 - val_loss: 0.1831 - val_accuracy: 0.9000
Epoch 28/100
13/12 [==============================] - ETA: -3s - loss: 0.2588 - accuracy: 0.8942
Epoch 00028: accuracy did not improve from 0.91106
13/12 [==============================] - 111s 9s/step - loss: 0.2588 - accuracy: 0.8942 - val_loss: 0.1942 - val_accuracy: 0.9200
```

```
13/12 [==============================] - 111s 9s/step - loss: 0.2588 - accuracy: 0.8942 - val_loss: 0.1942 - val_accuracy: 0.9200
Epoch 29/100
13/12 [==============================] - ETA: -2s - loss: 0.2045 - accuracy: 0.9111
Epoch 00029: accuracy did not improve from 0.91106
13/12 [==============================] - 108s 8s/step - loss: 0.2045 - accuracy: 0.9111 - val_loss: 0.1909 - val_accuracy: 0.9200
Epoch 30/100
13/12 [==============================] - ETA: -3s - loss: 0.1757 - accuracy: 0.9207
Epoch 00030: accuracy improved from 0.91106 to 0.92067, saving model to ./best_weights.hdf5
13/12 [==============================] - 173s 13s/step - loss: 0.1757 - accuracy: 0.9207 - val_loss: 0.1667 - val_accuracy: 0.9150
Epoch 31/100
13/12 [==============================] - ETA: -2s - loss: 0.2330 - accuracy: 0.9135
Epoch 00031: accuracy did not improve from 0.92067
13/12 [==============================] - 108s 8s/step - loss: 0.2330 - accuracy: 0.9135 - val_loss: 0.1615 - val_accuracy: 0.9200
Epoch 32/100
13/12 [==============================] - ETA: -3s - loss: 0.2193 - accuracy: 0.9207
Epoch 00032: accuracy did not improve from 0.92067
13/12 [==============================] - 110s 8s/step - loss: 0.2193 - accuracy: 0.9207 - val_loss: 0.1647 - val_accuracy: 0.9300
Epoch 33/100
13/12 [==============================] - ETA: -2s - loss: 0.2232 - accuracy: 0.9111
Epoch 00033: accuracy did not improve from 0.92067
13/12 [==============================] - 107s 8s/step - loss: 0.2232 - accuracy: 0.9111 - val_loss: 0.1793 - val_accuracy: 0.9200
Epoch 34/100
13/12 [==============================] - ETA: -2s - loss: 0.2365 - accuracy: 0.9062
Epoch 00034: accuracy did not improve from 0.92067
13/12 [==============================] - 106s 8s/step - loss: 0.2365 - accuracy: 0.9062 - val_loss: 0.1695 - val_accuracy: 0.9350
Epoch 35/100
13/12 [==============================] - ETA: -3s - loss: 0.2139 - accuracy: 0.9111
Epoch 00035: accuracy did not improve from 0.92067
13/12 [==============================] - 116s 9s/step - loss: 0.2139 - accuracy: 0.9111 - val_loss: 0.1644 - val_accuracy: 0.9300
Epoch 36/100
13/12 [==============================] - ETA: -2s - loss: 0.2252 - accuracy: 0.8942
Epoch 00036: accuracy did not improve from 0.92067
13/12 [==============================] - 104s 8s/step - loss: 0.2252 - accuracy: 0.8942 - val_loss: 0.1992 - val_accuracy: 0.9050
Epoch 37/100
13/12 [==============================] - ETA: -2s - loss: 0.2583 - accuracy: 0.8942
Epoch 00037: accuracy did not improve from 0.92067
13/12 [==============================] - 108s 8s/step - loss: 0.2583 - accuracy: 0.8942 - val_loss: 0.1680 - val_accuracy: 0.9250
Epoch 38/100
13/12 [==============================] - ETA: -2s - loss: 0.2024 - accuracy: 0.9351
Epoch 00038: accuracy improved from 0.92067 to 0.93510, saving model to ./best_weights.hdf5
13/12 [==============================] - 128s 10s/step - loss: 0.2024 - accuracy: 0.9351 - val_loss: 0.1705 - val_accuracy: 0.9250
Epoch 39/100
13/12 [==============================] - ETA: -2s - loss: 0.1600 - accuracy: 0.9399
Epoch 00039: accuracy improved from 0.93510 to 0.93990, saving model to ./best_weights.hdf5
13/12 [==============================] - 125s 10s/step - loss: 0.1600 - accuracy: 0.9399 - val_loss: 0.1672 - val_accuracy: 0.9200
Epoch 40/100
13/12 [==============================] - ETA: -3s - loss: 0.2066 - accuracy: 0.9111
Epoch 00040: accuracy did not improve from 0.93990
13/12 [==============================] - 111s 9s/step - loss: 0.2066 - accuracy: 0.9111 - val_loss: 0.1557 - val_accuracy: 0.9250
Epoch 41/100
13/12 [==============================] - ETA: -2s - loss: 0.2366 - accuracy: 0.8990
Epoch 00041: accuracy did not improve from 0.93990
13/12 [==============================] - 108s 8s/step - loss: 0.2366 - accuracy: 0.8990 - val_loss: 0.1422 - val_accuracy: 0.9250
Epoch 42/100
13/12 [==============================] - ETA: -2s - loss: 0.1580 - accuracy: 0.9327
```

```
13/12 [==============================] - 108s 8s/step - loss: 0.2366 - accuracy: 0.8990 - val_loss: 0.1422 - val_accuracy: 0.9250
Epoch 42/100
13/12 [==============================] - ETA: -2s - loss: 0.1580 - accuracy: 0.9327
Epoch 00042: accuracy did not improve from 0.93990
13/12 [==============================] - 108s 8s/step - loss: 0.1580 - accuracy: 0.9327 - val_loss: 0.1400 - val_accuracy: 0.9300
Epoch 43/100
13/12 [==============================] - ETA: -2s - loss: 0.2473 - accuracy: 0.8918
Epoch 00043: accuracy did not improve from 0.93990
13/12 [==============================] - 107s 8s/step - loss: 0.2473 - accuracy: 0.8918 - val_loss: 0.1402 - val_accuracy: 0.9400
Epoch 44/100
13/12 [==============================] - ETA: -2s - loss: 0.1852 - accuracy: 0.9087
Epoch 00044: accuracy did not improve from 0.93990
13/12 [==============================] - 104s 8s/step - loss: 0.1852 - accuracy: 0.9087 - val_loss: 0.1615 - val_accuracy: 0.9300
Epoch 45/100
13/12 [==============================] - ETA: -3s - loss: 0.1730 - accuracy: 0.9351
Epoch 00045: accuracy did not improve from 0.93990
13/12 [==============================] - 115s 9s/step - loss: 0.1730 - accuracy: 0.9351 - val_loss: 0.1654 - val_accuracy: 0.9300
Epoch 46/100
13/12 [==============================] - ETA: -2s - loss: 0.2143 - accuracy: 0.9111
Epoch 00046: accuracy did not improve from 0.93990
13/12 [==============================] - 103s 8s/step - loss: 0.2143 - accuracy: 0.9111 - val_loss: 0.1622 - val_accuracy: 0.9300
Epoch 47/100
13/12 [==============================] - ETA: -2s - loss: 0.1943 - accuracy: 0.9135
Epoch 00047: accuracy did not improve from 0.93990
13/12 [==============================] - 103s 8s/step - loss: 0.1943 - accuracy: 0.9135 - val_loss: 0.1829 - val_accuracy: 0.9300
Epoch 48/100
13/12 [==============================] - ETA: -2s - loss: 0.1970 - accuracy: 0.9183
Epoch 00048: accuracy did not improve from 0.93990
13/12 [==============================] - 104s 8s/step - loss: 0.1970 - accuracy: 0.9183 - val_loss: 0.1822 - val_accuracy: 0.9200
Epoch 49/100
13/12 [==============================] - ETA: -3s - loss: 0.2309 - accuracy: 0.9159
Epoch 00049: accuracy did not improve from 0.93990
13/12 [==============================] - 110s 8s/step - loss: 0.2309 - accuracy: 0.9159 - val_loss: 0.1614 - val_accuracy: 0.9250
Epoch 50/100
13/12 [==============================] - ETA: -2s - loss: 0.1769 - accuracy: 0.9231
Epoch 00050: accuracy did not improve from 0.93990
13/12 [==============================] - 105s 8s/step - loss: 0.1769 - accuracy: 0.9231 - val_loss: 0.1668 - val_accuracy: 0.9200
Epoch 51/100
13/12 [==============================] - ETA: -2s - loss: 0.2572 - accuracy: 0.8942
Epoch 00051: accuracy did not improve from 0.93990
13/12 [==============================] - 102s 8s/step - loss: 0.2572 - accuracy: 0.8942 - val_loss: 0.1599 - val_accuracy: 0.9300
Epoch 52/100
13/12 [==============================] - ETA: -2s - loss: 0.2131 - accuracy: 0.9014
Epoch 00052: accuracy did not improve from 0.93990
13/12 [==============================] - 101s 8s/step - loss: 0.2131 - accuracy: 0.9014 - val_loss: 0.1450 - val_accuracy: 0.9400
Epoch 53/100
13/12 [==============================] - ETA: -2s - loss: 0.1611 - accuracy: 0.9255
Epoch 00053: accuracy did not improve from 0.93990
13/12 [==============================] - 107s 8s/step - loss: 0.1611 - accuracy: 0.9255 - val_loss: 0.1389 - val_accuracy: 0.9450
Epoch 54/100
13/12 [==============================] - ETA: -2s - loss: 0.1978 - accuracy: 0.9183
Epoch 00054: accuracy did not improve from 0.93990
13/12 [==============================] - 103s 8s/step - loss: 0.1978 - accuracy: 0.9183 - val_loss: 0.1382 - val_accuracy: 0.9500
Epoch 00054: early stopping
```

Fig-5.5: DenseNet169 Implementation

VGG19

```
history1 = model1.fit(train_aug.flow(X_train, y_train, batch_size=64),
                      validation_data=(X_test, y_test),
                      validation_steps=len(X_test) / 64,
                      steps_per_epoch=len(X_train) / 64,
                      epochs=100,
                        callbacks= callback_list2
                     )
```

```
Epoch 1/100
13/12 [==============================] - ETA: -10s - loss: 1.0601 - accuracy: 0.6100
Epoch 00001: accuracy improved from -inf to 0.61000, saving model to Weights_AD_Model_VGG19.hdf5
13/12 [==============================] - 367s 28s/step - loss: 1.0601 - accuracy: 0.6100 - val_loss: 0.4452 - val_accuracy: 0.8050
Epoch 2/100
13/12 [==============================] - ETA: -9s - loss: 0.4782 - accuracy: 0.7650
Epoch 00002: accuracy improved from 0.61000 to 0.76500, saving model to Weights_AD_Model_VGG19.hdf5
13/12 [==============================] - 331s 25s/step - loss: 0.4782 - accuracy: 0.7650 - val_loss: 0.3645 - val_accuracy: 0.8450
Epoch 3/100
13/12 [==============================] - ETA: -9s - loss: 0.4130 - accuracy: 0.8250
Epoch 00003: accuracy improved from 0.76500 to 0.82500, saving model to Weights_AD_Model_VGG19.hdf5
13/12 [==============================] - 331s 25s/step - loss: 0.4130 - accuracy: 0.8250 - val_loss: 0.3309 - val_accuracy: 0.8600
Epoch 4/100
13/12 [==============================] - ETA: -10s - loss: 0.3802 - accuracy: 0.8275
Epoch 00004: accuracy improved from 0.82500 to 0.82750, saving model to Weights_AD_Model_VGG19.hdf5
13/12 [==============================] - 338s 26s/step - loss: 0.3802 - accuracy: 0.8275 - val_loss: 0.4496 - val_accuracy: 0.8100
Epoch 5/100
13/12 [==============================] - ETA: -10s - loss: 0.3572 - accuracy: 0.8487
Epoch 00005: accuracy improved from 0.82750 to 0.84875, saving model to Weights_AD_Model_VGG19.hdf5
13/12 [==============================] - 349s 27s/step - loss: 0.3572 - accuracy: 0.8487 - val_loss: 0.2968 - val_accuracy: 0.8600
Epoch 6/100
13/12 [==============================] - ETA: -10s - loss: 0.3645 - accuracy: 0.8438
Epoch 00006: accuracy did not improve from 0.84875
13/12 [==============================] - 346s 27s/step - loss: 0.3645 - accuracy: 0.8438 - val_loss: 0.3155 - val_accuracy: 0.8650
Epoch 7/100
13/12 [==============================] - ETA: -10s - loss: 0.3887 - accuracy: 0.8288
Epoch 00007: accuracy did not improve from 0.84875
13/12 [==============================] - 351s 27s/step - loss: 0.3887 - accuracy: 0.8288 - val_loss: 0.3240 - val_accuracy: 0.8650
Epoch 8/100
13/12 [==============================] - ETA: -10s - loss: 0.3429 - accuracy: 0.8475
Epoch 00008: accuracy did not improve from 0.84875
13/12 [==============================] - 333s 26s/step - loss: 0.3429 - accuracy: 0.8475 - val_loss: 0.2769 - val_accuracy: 0.8700
Epoch 9/100
13/12 [==============================] - ETA: -10s - loss: 0.3397 - accuracy: 0.8313
Epoch 00009: accuracy did not improve from 0.84875
13/12 [==============================] - 339s 26s/step - loss: 0.3397 - accuracy: 0.8313 - val_loss: 0.2957 - val_accuracy: 0.8700
Epoch 10/100
13/12 [==============================] - ETA: -10s - loss: 0.3160 - accuracy: 0.8537
Epoch 00010: accuracy improved from 0.84875 to 0.85375, saving model to Weights_AD_Model_VGG19.hdf5
13/12 [==============================] - 349s 27s/step - loss: 0.3160 - accuracy: 0.8537 - val_loss: 0.2538 - val_accuracy: 0.9000
Epoch 11/100
13/12 [==============================] - ETA: -11s - loss: 0.3504 - accuracy: 0.8425
Epoch 00011: accuracy did not improve from 0.85375
13/12 [==============================] - 367s 28s/step - loss: 0.3504 - accuracy: 0.8425 - val_loss: 0.2444 - val_accuracy: 0.8950
Epoch 12/100
13/12 [==============================] - ETA: -10s - loss: 0.3453 - accuracy: 0.8413
Epoch 00012: accuracy did not improve from 0.85375
13/12 [==============================] - 349s 27s/step - loss: 0.3453 - accuracy: 0.8413 - val_loss: 0.3495 - val_accuracy: 0.8400
Epoch 13/100
13/12 [==============================] - ETA: -10s - loss: 0.3169 - accuracy: 0.8612
Epoch 00013: accuracy improved from 0.85375 to 0.86125, saving model to Weights_AD_Model_VGG19.hdf5
13/12 [==============================] - 347s 27s/step - loss: 0.3169 - accuracy: 0.8612 - val_loss: 0.2976 - val_accuracy: 0.8650
Epoch 14/100
13/12 [==============================] - ETA: -10s - loss: 0.3287 - accuracy: 0.8450
Epoch 00014: accuracy did not improve from 0.86125
13/12 [==============================] - 347s 27s/step - loss: 0.3287 - accuracy: 0.8450 - val_loss: 0.3243 - val_accuracy: 0.8600
```

```
Epoch 00049: accuracy did not improve from 0.92750
13/12 [==============================] - 366s 28s/step - loss: 0.2127 - accuracy: 0.9013 - val_loss: 0.1968 - val_accuracy: 0.9200
Epoch 50/100
13/12 [==============================] - ETA: -11s - loss: 0.2188 - accuracy: 0.9162
Epoch 00050: accuracy did not improve from 0.92750
13/12 [==============================] - 368s 28s/step - loss: 0.2188 - accuracy: 0.9162 - val_loss: 0.1874 - val_accuracy: 0.9350
Epoch 51/100
13/12 [==============================] - ETA: -10s - loss: 0.2066 - accuracy: 0.9100
Epoch 00051: accuracy did not improve from 0.92750
13/12 [==============================] - 363s 28s/step - loss: 0.2066 - accuracy: 0.9100 - val_loss: 0.2155 - val_accuracy: 0.9400
Epoch 52/100
13/12 [==============================] - ETA: -11s - loss: 0.2241 - accuracy: 0.9100
Epoch 00052: accuracy did not improve from 0.92750
13/12 [==============================] - 372s 29s/step - loss: 0.2241 - accuracy: 0.9100 - val_loss: 0.2602 - val_accuracy: 0.8800
Epoch 53/100
13/12 [==============================] - ETA: -11s - loss: 0.2774 - accuracy: 0.8863
Epoch 00053: accuracy did not improve from 0.92750
13/12 [==============================] - 392s 30s/step - loss: 0.2774 - accuracy: 0.8863 - val_loss: 0.3120 - val_accuracy: 0.8850
Epoch 54/100
13/12 [==============================] - ETA: -10s - loss: 0.2446 - accuracy: 0.9013
Epoch 00054: accuracy did not improve from 0.92750
13/12 [==============================] - 350s 27s/step - loss: 0.2446 - accuracy: 0.9013 - val_loss: 0.1844 - val_accuracy: 0.9350
Epoch 55/100
13/12 [==============================] - ETA: -10s - loss: 0.2125 - accuracy: 0.9062
Epoch 00055: accuracy did not improve from 0.92750
13/12 [==============================] - 354s 27s/step - loss: 0.2125 - accuracy: 0.9062 - val_loss: 0.1839 - val_accuracy: 0.9300
Epoch 56/100
13/12 [==============================] - ETA: -11s - loss: 0.2098 - accuracy: 0.9075
Epoch 00056: accuracy did not improve from 0.92750
13/12 [==============================] - 366s 28s/step - loss: 0.2098 - accuracy: 0.9075 - val_loss: 0.1820 - val_accuracy: 0.9300
Epoch 57/100
13/12 [==============================] - ETA: -11s - loss: 0.2166 - accuracy: 0.9038
Epoch 00057: accuracy did not improve from 0.92750
13/12 [==============================] - 383s 29s/step - loss: 0.2166 - accuracy: 0.9038 - val_loss: 0.1838 - val_accuracy: 0.9200
Epoch 58/100
13/12 [==============================] - ETA: -10s - loss: 0.2104 - accuracy: 0.9000
Epoch 00058: accuracy did not improve from 0.92750
13/12 [==============================] - 356s 27s/step - loss: 0.2104 - accuracy: 0.9000 - val_loss: 0.1896 - val_accuracy: 0.9400
Epoch 59/100
13/12 [==============================] - ETA: -10s - loss: 0.2391 - accuracy: 0.8900
Epoch 00059: accuracy did not improve from 0.92750
13/12 [==============================] - 364s 28s/step - loss: 0.2391 - accuracy: 0.8900 - val_loss: 0.2668 - val_accuracy: 0.8750
Epoch 60/100
13/12 [==============================] - ETA: -11s - loss: 0.1979 - accuracy: 0.9237
Epoch 00060: accuracy did not improve from 0.92750
13/12 [==============================] - 367s 28s/step - loss: 0.1979 - accuracy: 0.9237 - val_loss: 0.2168 - val_accuracy: 0.9300
Epoch 61/100
13/12 [==============================] - ETA: -11s - loss: 0.2640 - accuracy: 0.8963
Epoch 00061: accuracy did not improve from 0.92750
13/12 [==============================] - 380s 29s/step - loss: 0.2640 - accuracy: 0.8963 - val_loss: 0.3474 - val_accuracy: 0.8500
Epoch 62/100
13/12 [==============================] - ETA: -10s - loss: 0.3184 - accuracy: 0.8737
Epoch 00062: accuracy did not improve from 0.92750
13/12 [==============================] - 369s 28s/step - loss: 0.3184 - accuracy: 0.8737 - val_loss: 0.3076 - val_accuracy: 0.8900
Epoch 00062: early stopping
```

Fig-5.6: VGG19 Implementation

5.3 Testing and Results

5.3.1 Confusion matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

True Positive (TP)

• The predicted value matches the actual value.

• The actual value was positive, and the model predicted a positive value.

True Negative (TN)

• The predicted value matches the actual value.

• The actual value was negative, and the model predicted a negative value.

False Positive (FP) – Type 1 error

• The predicted value was falsely predicted.

• The actual value was negative, but the model predicted a positive value.

• Also known as the Type 1 error

False Negative (FN) – Type 2 error

• The predicted value was falsely predicted.

• The actual value was positive, but the model predicted a negative value.

• Also known as the Type 2 error.

DenseNet169

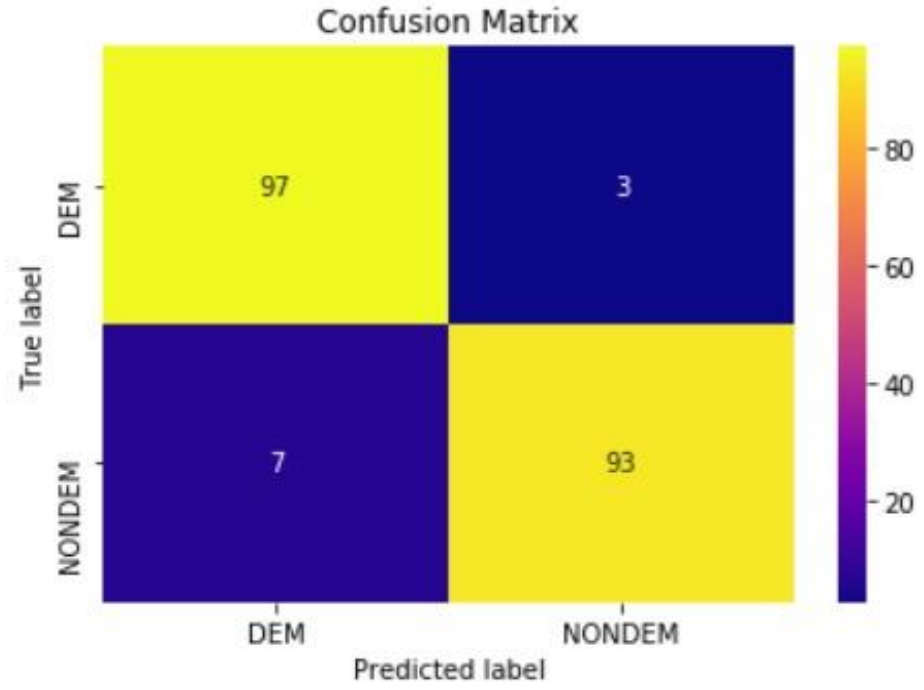

Fig-5.7: Confusion Matrix of DenseNet169

VGG19

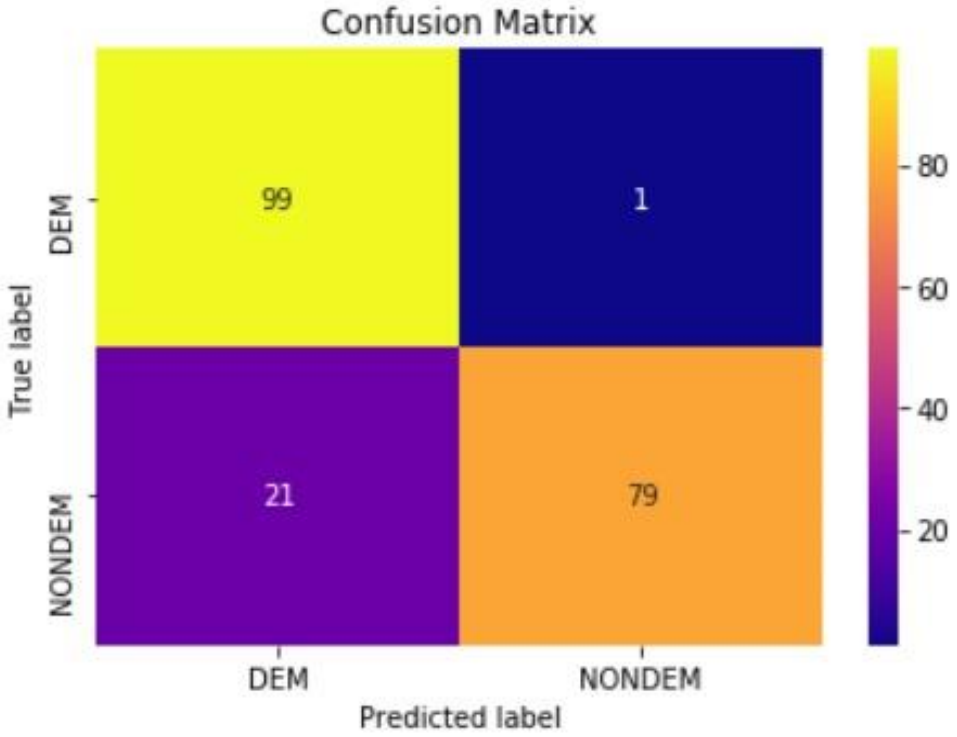

Fig-5.8: Confusion Matrix of VGG19

5.3.2 ROC Curve

DenseNet169

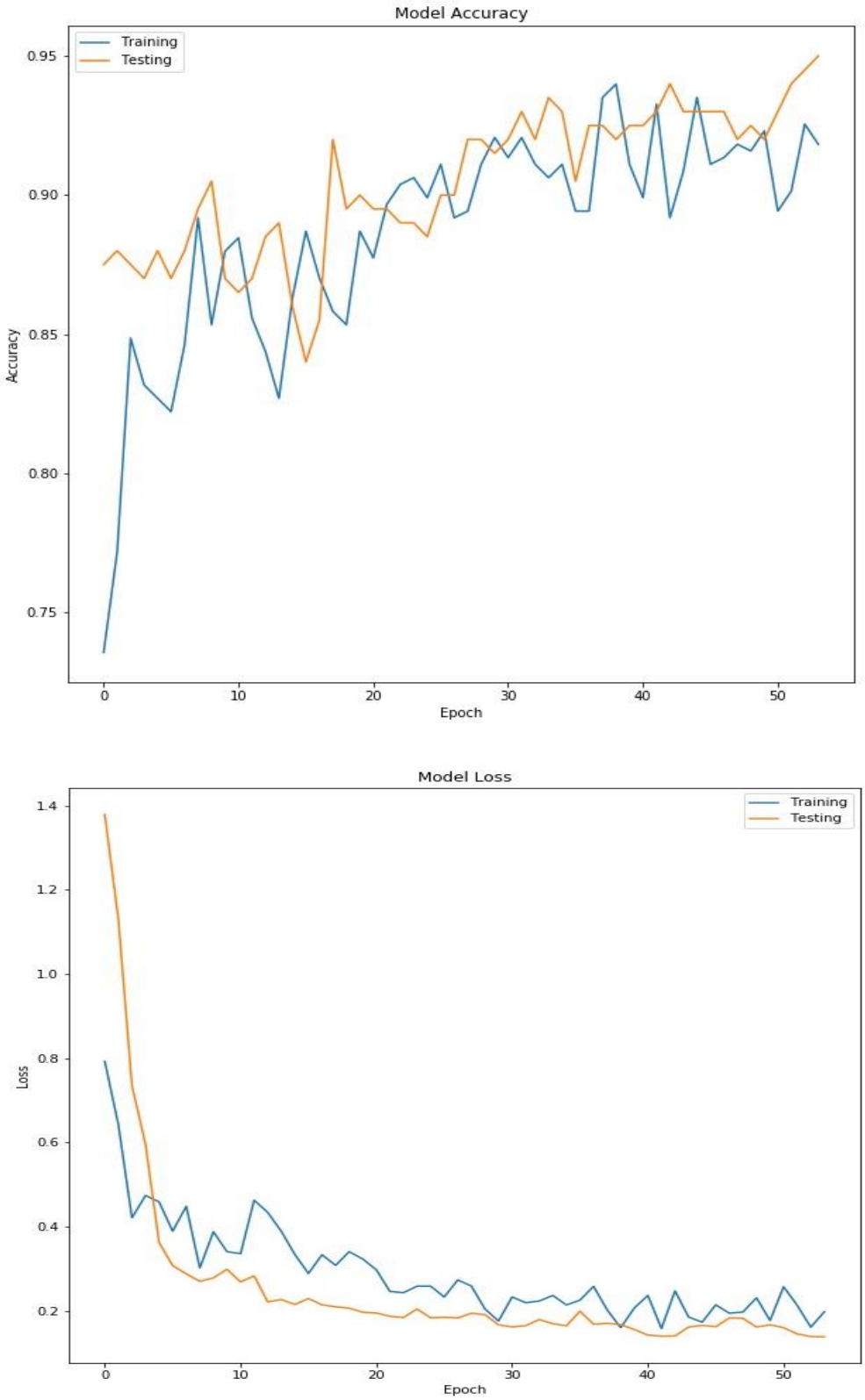

Fig-5.9: ROC curves for DenseNet169

VGG19

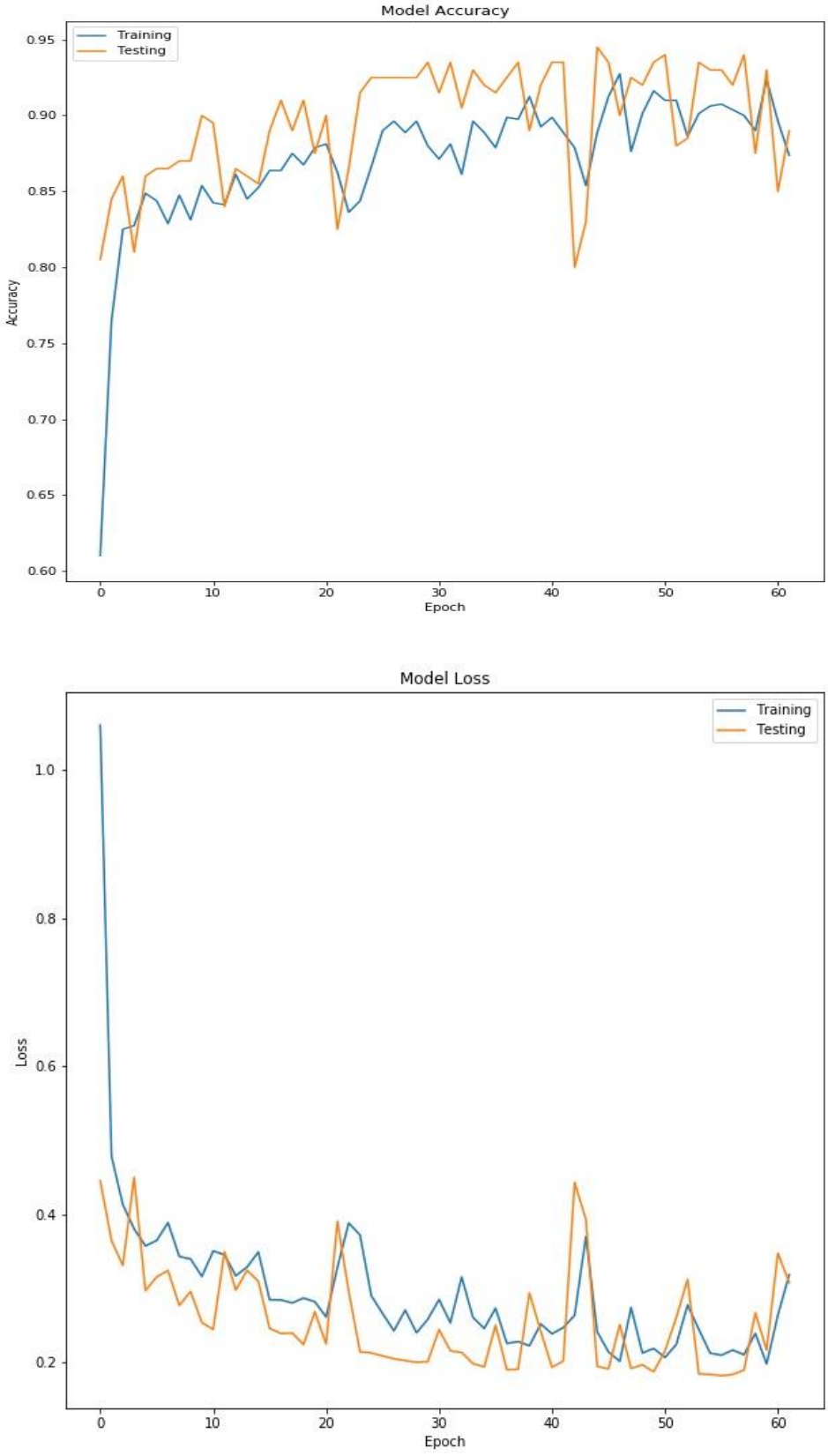

Fig-5.10: ROC curves for VGG19

5.3.3 Classification Report

Evaluation of the system can be done using following methods:

a) Precision: Precision estimates the predictive value of a label, either positive or negative, depending on the class for which it is calculated; in other words, it assesses the predictive power of the algorithm. Precision is the percentage of correctly assigned expressions in relation to the total number of aspects.

$$precision = \frac{tp}{(tp + fp)}$$

b) Recall: Recall is a function of its correctly classified examples (true positives) and its misclassified examples (false negatives). Recall is the percentage of correctly assigned expressions in relation to the total number of expressions.

$$recall = \frac{tp}{p + fn}$$

c) F1-score: F-score is a composite measure which benefits algorithms with higher sensitivity and challenges algorithms with higher specificity. The F-score is evenly balanced when β = 1. It favours precision when β > 1 and recall otherwise.

$$\mathrm{F} - \text{measure} = \frac{(\beta\, 2 + 1) * \text{precision} * recall}{\beta 2 * \text{precision} * \text{recall}}$$

All three measures distinguish the correct classification of labels within different classes. They concentrate on one class (positive examples). Hence, precision and recall do measure different 27 properties and we therefore need a combined quality measure in order to determine the best matching aspect to expression category mappings. The so-called F measure computes the harmonic mean of precision and recall and allows considering both properties at the same time. The overall recall is also known as accuracy.

DenseNet169

	precision	recall	f1-score	support
0	0.93	0.97	0.95	100
1	0.97	0.93	0.95	100
accuracy			0.95	200
macro avg	0.95	0.95	0.95	200
weighted avg	0.95	0.95	0.95	200

Fig-5.11: Classification report of DenseNet169

VGG19

	precision	recall	f1-score	support
0	0.82	0.99	0.90	100
1	0.99	0.79	0.88	100
accuracy			0.89	200
macro avg	0.91	0.89	0.89	200
weighted avg	0.91	0.89	0.89	200

Fig-5.12: Classification report of VGG19

CHAPTER-6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

Alzheimer's disease is the leading cause of dementia. Subtle and spatially complex deformation patterns of hippocampus between patients with AD and healthy control subjects can be detected by machine learning methods. This project determines a prospective solution for detecting the disease at an early stage. The models used in this project have successfully classified the images into the appropriate two classes and indeed provided us with promising results. We observe that DenseNet performs better than VGG19. Further research is required to ensure that this particular model can be implemented in clinical settings, increasing the health care rate against this specific disease. Knowledge should be spread among people regarding this disease, and they should be encouraged to get themselves examined.

6.2 Future Scope

- We can deploy this model onto a website for better practical usage. In the future, this model can also be tested on a larger dataset.
- The proposed model can help doctors diagnose Alzheimer's Disease more effectively and can be modified to identify other neurodegenerative diseases more automatically in the future.
- It can be built into a proper user interface application so that it is easily accessible to everyone.
- It can be transferred as a tool in hospitals so that it helps the patients with Alzheimer’s disease.

REFERENCES

[1] World Health Organization. World health statistics 2010. World Health Organization,2010.

[2] Jin K, Simpkins JW, Ji X, Leis M, Stambler I. The critical need to promote research of aging and aging-related diseases to improve health and longevity of the elderly population. Aging and disease. 2015

[3] Khan, Afreen, and Swaleha Zubair. "An Improved Multi-Modal based Machine Learning Approach for the Prognosis of Alzheimer's disease." Journal of King Saud University-Computer and Information Sciences, 2020.

[4] Khan, Afreen, and Swaleha Zubair. "Usage Of Random Forest Ensemble Classifier Based Imputation and Its Potential In The Diagnosis Of Alzheimer's Disease." Int. J. Sci. Technol. Res. 8, no. 12, 2019, pp. 271- 275.

[5] Asim, Yousra, Basit Raza, Ahmad Kamran Malik, Saima Rathore, Lal Hussain, and Mohammad Aksam Iftikhar. "A multi-modal, multi-atlas- based approach for Alzheimer detection via machine learning." International Journal of Imaging Systems and Technology 28, no. 2, 2018, pp. 113-123.

[6] Alam, Saruar, Goo-Rak Kwon, and Alzheimer's Disease Neuroimaging Initiative. "Alzheimer disease classification using KPCA, LDA, and multi-kernel learning SVM." International Journal of Imaging Systems and Technology 27, no. 2, 2017, pp. 133-143

[7] Khajehnejad, Moein, Forough Habibollahi Saatlou, and Hoda Mohammadzade. "Alzheimer's disease early diagnosis using manifoldbased semi-supervised learning." Brain sciences 7, no. 8, 2017, p. 109.

[8] Lama, Ramesh Kumar, Jeonghwan Gwak, Jeong-Seon Park, and SangWoong Lee. "Diagnosis of Alzheimer's disease based on structural MRI images using a regularized extreme learning machine and PCA features." Journal of healthcare engineering 2017, 2017.

[9] Bryan, R. Nick. "Machine learning applied to Alzheimer disease.", 2016, pp. 665-668.

[10] Escudero, Javier, Emmanuel Ifeachor, John P. Zajicek, Colin Green, James Shearer, Stephen Pearson, and Alzheimer's Disease Neuroimaging Initiative. "Machine learning-based method for personalized and costeffective detection of Alzheimer's disease." IEEE transactions on biomedical engineering 60, no. 1, 2012, pp. 164-168

[11] Aruchamy S, Kumar RK, Bhattacharjee P, Sanyal G. Automated skull stripping in brain MR images. In2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom) 2016 Mar 16 (pp. 2043-2047). IEEE

[12] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.