DEPLOYABLE AND WEIGHTED ENSEMBLE-BASED DEEP LEARNING MODEL FOR ALZHEIMER'S DISEASE DETECTION

A PROJECT REPORT

Submitted By

SUNDEEP S 185001176

VEERARAGHAVAN N 185001191

VISHAL N 185001198

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Rajiv Gandhi Salai (OMR), Kalavakkam - 603110

May 2022

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

BONAFIDE CERTIFICATE

Certified that this project report titled **"DEPLOYABLE AND WEIGHTED ENSEMBLE-BASED DEEP LEARNING MODEL FOR ALZHEIMER'S DISEASE DETECTION"** is the *bonafide* work of "**SUNDEEP S (185001176)**, **VEERARAGHAVAN N (185001191)**, and **VISHAL N (185001198)**" who carried out the project work under my supervision.

Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. T.T. Mirnalinee
Head of the Department
Professor,
Department of CSE,
SSN College of Engineering,
Kalavakkam - 603 110

Dr. Y. V. Lokeswari
Supervisor
Associate Professor,
Department of CSE,
SSN College of Engineering,
Kalavakkam - 603 110

Place:
Date:

Submitted for the examination held on. . . . . .

Internal Examiner **External Examiner**

ACKNOWLEDGEMENTS

I thank GOD, the almighty for giving me strength and knowledge to do this project.

I would like to thank and deep sense of gratitude to my guide **Dr. Y. V. LOKESWARI**, Associate Professor, Department of Computer Science and Engineering, for his valuable advice and suggestions as well as his continued guidance, patience and support that helped me to shape and refine my work.

My sincere thanks to **Dr. T.T. MIRNALINEE**, Professor and Head of the Department of Computer Science and Engineering, for her words of advice and encouragement and I would like to thank our project Coordinator **Dr. B. BHARATHI**, Associate Professor, Department of Computer Science and Engineering for her valuable suggestions throughout this project. I also express my gratitude to the panel members **Dr. D. VENKATA VARA PRASAD**, **Dr. J. SURESH**, and **DR. S. V. JANSI RANI**.

I express my deep respect to the founder **Dr. SHIV NADAR**, Chairman, SSN Institutions. I also express my appreciation to our **Dr. V. E. ANNAMALAI**, Principal, for all the help he has rendered during this course of study. Furthermore, I would like to extend my appreciation to the management for providing facilities that made this project possible.

I would like to extend my sincere thanks to all the teaching and non-teaching staffs of our department who have contributed directly and indirectly during the course of my project work. Finally, I would like to thank my parents and friends for their patience, cooperation and moral support throughout my life.

SUNDEEP S **VEERARAGHAVAN N** **VISHAL N**

ABSTRACT

Deep Learning is a subset of Artificial Intelligence (AI) and Machine Learning (ML) which can understand and learn hidden features in images and make better predictions. Healthcare field is increasingly seeing automated AI involvement in making disease predictions by providing sufficient labelled data. Alzheimer's disease is a neurological condition in which the death of brain cells causes memory loss and cognitive decline. In real-world, multiple doctors classify a subject's case as Alzheimer's Disease (AD) or Normal Cognition (NC) and the true classification is taken by consensus of all the doctors. In a similar fashion, this project aims to train multiple Deep Learning models, create an ensemble and make a master multi-class prediction by concatenating the outputs of all the individual classifiers. Currently, most Deep Learning models for Alzheimer's Disease prediction use cloud server architecture which increases the inference time manifold and opens up the risk of data breach as personal data of subjects are sent to the cloud for computation. Therefore, this work intends to test and record the inference time of our Alzheimer's Disease architecture by deploying it on Raspberry Pi microprocessor. The proposed model achieved 19% higher accuracy and 6.5% higher sensitivity than existing models with an inference time of 1.97 seconds when deployed on Raspberry Pi.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Alzheimer's disease is one of the most devastating brain disorders in elderly humans. It is an under-treated and under-recognized disease that is becoming a major public health problem [18]. More than 25 million people in the world today are affected by dementia, mostly suffering from Alzheimer's disease. In both developed and developing nations, Alzheimer's disease has had tremendous impact on the affected individuals, caregivers, and society [16]. According to World Health Organization (WHO), in 2019, the estimated total global societal cost of dementia was US $1.3 trillion, and these costs are expected to surpass US $2.8 trillion by 2030 as both the number of people living with dementia and care costs increase [25]. Evidently, dementia has significant social and economic impact over society and state. Abnormalities in brain tend to adversely affect the neurocognitive abilities of human beings, in most cases, permanently. Therefore, there is a pressing need for detection of Alzheimer's disease at the earliest.

1.1 ALZHEIMER'S DISEASE CAUSE AND EFFECT

Alzheimer's disease is a progressive neurological disorder that causes the brain to shrink (atrophy) and brain cells to die. It is the most common cause of dementia — a continuous decline in thinking, behavioral and social skills that affects a person's ability to function independently. The exact causes of

Alzheimer's disease are not fully understood. But at a basic level, brain proteins fail to function normally, which disrupts the work of brain cells (neurons) and triggers a series of toxic events. This damages the neurons and their connections to each other thereby reducing the brain function drastically. The early signs of the disease include forgetting recent events or conversations. As the disease progresses, a person with Alzheimer's disease will develop severe memory impairment and lose the ability to carry out everyday tasks. Currently, there is no treatment that cures Alzheimer's disease or alters the disease process in the brain. In advanced stages of the disease, complications from severe loss of brain function —- such as dehydration, malnutrition or infection —- result in death.

1.2 SUPERVISED AND UNSUPERVISED LEARNING

Supervised learning is a learning in which the machine is taught or trained using data which is well labeled. After that, the machine is provided with a new set of examples (data) so that supervised learning algorithm analyses the training data (set of training examples) and produces a correct outcome from labeled data. Supervised learning is classified into two categories of algorithms:

- **Classification**: A classification problem is when the output variable is a discrete category.
- **Regression**: A regression problem is when the output variable is a continuous real value.

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of the machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unsupervised learning is classified into two categories of algorithms:

- **Clustering**: A clustering problem discovers the inherent groupings lurking in the data.
- **Association**: An association rule learning problem discovers rules that describe large portions of the data.

1.3 DEEP LEARNING

Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as *deep neural networks*. Typical neural networks appear as illustrated in Fig. 1.1.

One of the most popular types of deep neural networks is known as Convolutional Neural Networks (CNN or ConvNet). A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images.

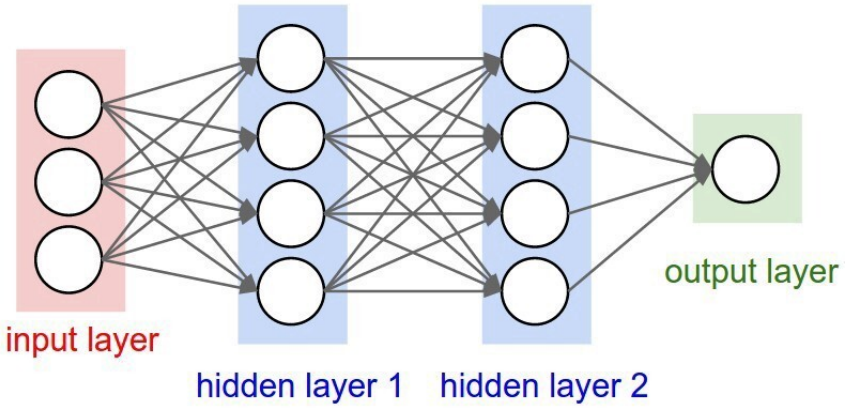

FIGURE 1.1: Neural Network

1.4 SUMMARY

This chapter gives an overview of the need for an automated Alzheimer's Disease detection, its cause and effect, and delves into the differences between supervised and unsupervised learning paradigms. This chapter ends with shedding light on the different deep learning architectures in specific reference to Convolutional Neural Networks.

The rest of the thesis is arranged as follows: Chapter 2 discusses some of the related works under the domain and Chapter 3 briefs the system requirements, Chapter 4 elucidates the design phase. The implementation phase is thoroughly written in Chapter 5. Chapter 6 proceeds to give an overview on result analysis. Chapter 7 summarizes the work, its limitations and the future enhancements possible.

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

This chapter discusses the literature studied and surveyed in order to conceptualize the project and define its objectives along with various implementation strategies.

2.2 LITERATURE RELATED TO ALZHEIMER'S DISEASE

2.2.1 A Comprehensive Machine-Learning Model Applied to Magnetic Resonance Imaging (MRI) to Predict Alzheimer's Disease (AD) in Older Subjects

Year & Publication: 2021 & Research Square

Technique: In this study, we present a sophisticated Machine Learning (ML) model of great accuracy to diagnose the early stages of AD. A total of 373 MRI tests belonging to 150 subjects (age $\geq$ 60) were examined and analysed in parallel with fourteen distinct features related to standard AD diagnosis. Four Machine Learning (ML) models, such as Naive Bayes (NB), Artificial Neural Networks (ANN), K-Nearest Neighbours (KNN), and Support-Vector Machines

(SVM), and the Receiver Operating Characteristic (ROC) curve metric were used to validate the model performance. [1]

Cons: Early Alzheimer's detection was not that successful with the small proprietary dataset. They had to use multiple methods to achieve acceptable accuracy, but pointless when the dataset itself is minuscule. The paper only focuses on increasing the accuracy rather than the sensitivity, which is a far more important metric.

2.2.2 Input Agnostic Deep Learning for Alzheimer's Disease Classification Using Multimodal MRI Images

Year & Publication: 2021 & Journal of Clinical Medicine

Technique: The data used in this work was extracted from the OASIS-3 dataset. This dataset includes MRI and PET images from 1098 subjects that were collected at the Washington University Knight Alzheimer Disease Research Center over 15 years. We used only T1w and DTI scans for the Normal Cognition (NC), Mild Cognitive Impairment (MCI), and Alzheimer's Disease (AD) classes. In this work, we first trained three networks (for T1w, FA, and MD scans) using the pretrained ResNet18 as the base to classify NC, MCI, and AD classes. After fine-tuning each network, three models with the best performance were selected to create the multi-modal network. [12]

Cons: The training was performed on a high-end Nvidia DGX-2 server. This increases cost of training and inference time if deployed in a real-world scenario. Another problem is that this paper focuses more on accuracy rather than sensitivity.

2.2.3 MRI Segmentation and Classification of Human Brain Using Deep Learning for Diagnosis of Alzheimer's Disease: A Survey

Year & Publication: 2021 & Electrical Engineering and Systems Science > Image and Video Processing

Technique: The data evaluation framework of three-dimension (3D) cross-sectional brain MRI is used to classify patients with AD and to segment brain tissue types (CerebroSpinal Fluid, Grey Matter, and White Matter). Publicly available datasets such as Open Access Series of Imaging Studies (OASIS), Alzheimer's Disease Neuroimaging Initiative (ADNI), Medical Image Computing and Computer-Assisted Intervention (MICCAI), and Internet Brain Segmentation Repository (IBSR) are popularly used for segmentation of brain MRI and AD diagnosis. [23]

Cons: One of the biggest limitations of this paper is that they use manual separation of MRI. This can be very time-consuming and requires in-depth knowledge of the anatomy of the brain. Furthermore, the inference time is high because of the use of highly complex CNNs.

2.2.4 Assisted Diagnosis of Alzheimer's Disease Based on Deep Learning and Multimodal Feature Fusion

Year & Publication: 2021 & Complexity

Technique: The amount of experiment data in this paper is very small; in order to avoid as much as possible the overfitting phenomenon that often occurs in CNNs, this paper uses a lightweight network ShuffleNet with fewer parameters and PCANet that does not require feedback adjustment parameters to implement deep features extraction and classification. [21]

Cons: Datasets including 34 cases of AD, 36 cases of MCI (including 18 cases of early MCI, 18 cases of late MCI), and 50 cases of NC were finally selected as experimental data. This dataset is very small, thus the model cannot be trusted for more varied input data.

2.2.5 Deep learning prediction of mild cognitive impairment conversion to Alzheimer's disease at 3 years after diagnosis using longitudinal and whole-brain 3D MRI

Year & Publication: 2021 & PeerJ

Technique: This retrospective study consisted of 320 normal cognition (NC), 554 MCI, and 237 AD patients. Longitudinal data include T1-weighted 3D MRI obtained at initial presentation with diagnosis of MCI and at 12-month follow up. Whole-brain 3D MRI volumes were used without a priori segmentation of regional structural volumes or cortical thicknesses. MRIs of the AD and NC cohort were used to train a deep learning classification model to obtain weights to be applied via transfer learning for prediction of MCI patient conversion to AD at three years post-diagnosis. Two (zero-shot and fine-tuning) transfer learning methods were evaluated. Three different Convolutional Neural Network (CNN)

architectures (sequential, residual bottleneck, and wide residual) were compared. Data were split into 75% and 25% for training and testing, respectively, with 4-fold cross validation. Prediction accuracy was evaluated using balanced accuracy. Heatmaps were generated. [13]

Cons: The dataset used is a proprietary one. The authors use multiple CNNs with different architecture. This makes it expensive to train and increases the inference time. The authors also focused more on accuracy, which in itself is a low 79 percent, rather than sensitivity.

2.2.6 Gerontechnology - The Study of Alzheimer's Disease using Cloud Computing

Year & Publication: 2017 & IEEE International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)

Technique: Cloud storage and processing section, fog computing layer and frontend data collection layer forms the architecture. The data obtained in the frontend layer is transmitted to the cloud servers through the fog layer. The dataset contains symptoms of patients along with their personal information. K-Means Clustering algorithm, which runs on a cloud server, is implemented to classify whether Alzheimer's disease is present or not. To evaluate the performance of the system, different experiments like Synthetic data generation, Fuzzy K-Nearest Neighbours (FKNN) and results from Decision Tree in WEKA tool were used. [19]

Cons: Sensitive data is sent to the cloud for processing without proper privacy measures. Cloud based training makes it expensive and increases the inference time.

2.2.7 Enhancing magnetic resonance imaging-driven Alzheimer's disease classification performance using Generative Adversarial learning

Year & Publication: 2021 & Springer

Technique: We obtained access to T1-weighted MRI scans from the ADNI (n = 417), AIBL (n = 107), and NACC (n = 565) cohorts. For a subset of the ADNI data (n = 151), both 1.5-Tesla (1.5-T) and 3-Tesla (3-T) scans taken at the same time were available, and 1.5-T scans were available from the other cohorts. All the MRI scans considered for this study were performed on individuals within $\pm$ 6 months from the date of clinical assessment. [24]

Cons: One limitation is that the sample size of the GAN network was very small (n = 151). Moreover, the models used are very intensive, thus further optimization can be done to reduce inference time.

2.3 LITERATURE RELATED TO RASPBERRYPI 4

2.3.1 Disease detection in tomato leaves via CNN with lightweight architectures implemented in Raspberry Pi 4

Year & Publication: 2021 & Computer and Electronics in Agriculture

Technique: Uses Transfer Learning of several models including MobileNetV2, MobileNetV3, Xception, etc. Finds that GoogLeNet, AlexNet, etc. have many millions of parameters, hence does not deploy. Includes quantitative metrics (accuracy, precision, recall, f1-score) and qualitative metric (saliency map). Includes a simple GUI at the end. Key takeways include the ability to deploy a model with 9 million parameters on Raspberry Pi 4. [5]

Cons: MobileNetV2 reports a low sensitivity score of 0.75, while Xception which performs the best has over 23 million parameters that is not suitable for edge settings.

2.3.2 Green Leaf Disease detection Using Raspberry Pi

Year & Publication: 2019 & 1st International Conference on Innovations in Information and Communication Technology (ICIICT)

Technique: The authors of this research paper use K-means Clustering Algorithm for image analysis of the green leaves for disease detection. They

perform basic OpenCV operations, and scikit-learn functions to train a basic algorithm. The architecture is connected wirelessly to a SIM and an email notification functionality is added. [11]

Cons: No scores are reported. Clustering model, in theory, should consume relatively large amount of memory and fits comfortably in a Raspberry Pi.

2.3.3 COVID-19 diagnosis from CT scans and chest X-ray images using low-cost Raspberry Pi

Year & Publication: 2021 & Plos One

Technique: Local features are extracted using Local Binary Pattern (LBP) algorithm. Global features are extracted from chest X-Rays and CT Scans. The proposed system steps are integrated to fit the low computational and memory capacities of the embedded system. The proposed method has the smallest computational and memory resources,less than the state-of-the-art methods by two to three orders of magnitude, among existing state-of-the-art deep learning (DL)-based methods. Uses the deep transfer learning approach with four architectures, namely, VGG-19, MobileNet, Inception, and Xception. Their proposed method has the highest accuracy (98.75%) using the VGG-19 model. [7]

2.4 LITERATURE RELATED TO ENSEMBLE NETWORKS

2.4.1 An Ensemble of Deep Convolutional Neural Networks for Alzheimer's Disease Detection and Classification

Year & Publication: 2017 & Neural Information Processing Systems

Technique: The data used in this work is Open Access Series of Imaging Studies (OASIS) dataset. For each MRI data, patches have been created from three physical planes of imaging: Axial or horizontal plane, Coronal or frontal plane, and Sagittal or median plane. This model is an ensemble of 3 DenseNet styled models - DenseNet-121, DenseNet-161, and DenseNet-169. These 3 models have been pre-trained with ImageNet dataset. The individual models take an MRI image as input and generates its learned representation. Based on this feature representation, the input MRI image is classified to any of the four output classes. The output classification label of the three individual models are ensembled together using voting technique. Each classifier "votes" for a particular class, and the class with the majority votes would be assigned as the label for the input MRI data. Since the dataset is small, 5-fold cross validation is performed on the dataset. For each fold, We have used 70% as training data, 10% as validation data and 20% as test data. The individual models are optimized with the Stochastic Gradient Descent (SGD) algorithm and early-stopping is used for regularization. The accuracy of the proposed model is 93.18% with 93% precision, 92% recall and 92% f1–score. [9]

2.4.2 Early Detection of Alzheimer's Disease using Magnetic Resonance Imaging: A Novel Approach Combining Convolutional Neural Networks and Ensemble Learning

Year & Publication: 2020 & Frontiers in Neuroscience

Technique: Data used in the study were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database. The preprocessing pipeline included skull extraction, registration to the MNI space, and image smoothing, so that after processing, all the images had a dimension of 121 * 145 * 121. Each base classifier consisted of six convolution layers (conv) and two fully connected layers (FCs). Phase 1 of the model involves building three classifier ensembles based on single-axis slices (i.e., sagittal, coronal, and transverse) and Phase 2 involves constructing a classifier ensemble based on three-axis slices. Then, the five base classifiers with the best generalization performance for each slice orientation were selected using the verification dataset. The output of a classifier ensemble based on single-axis slices was generated by combining the outputs of the five best base classifiers. Finally, a simple majority voting scheme was used to combine the predictions of these three classifier ensemble to yield the output of the classifier ensemble. [14]

Cons: The average classification accuracies were comparatively low with 84% for AD vs. HC, 79% for MCIc vs. HC, and 62% for MCIc vs. MCInc.

2.4.3 3D DenseNet Ensemble in 4-Way Classification of Alzheimer's Disease

Year & Publication: 2020 & Brain Informatics

Technique: MRI Data Structural brain MRI scans from the ADNI dataset were used (n=600 images) in this study. Preprocessed MRI scans(e.g., mask, intensity normalisation, reorientation, and spatial normalisation) were downloaded in NIfTI file format from ADNI2 and ADNIGO. For all the experiments the dataset was divided into 80% training and 20% validation. Multiple tests on the 3D DenseNet were carried out and the network hyper-parameters were optimised to obtain best results on the 4-way classification task. The proposed approach employs distinct 3D DenseNets that vary in their hyperparameters. These DenseNets are fed with MR images that pass through the network and the networks classification probability goes to a probability-based fusion approach to make the last classification. [17]

Cons: While comparing with other models which use 3D MRI as input, this proposed model achieved the 83.33%.

2.4.4 Automated Diagnosis of ear disease using ensemble deep learning with a big otoendoscopy image database

Year & Publication: 2019 & eBioMedicine

Technique: Image Data from patients who visited the outpatient clinic in Severance Hospital otorhinolaryngology department from the year 2013 to 2017

is used int this study. Totally, the dataset contains 10544 images. Convolution-based deep neural networks like Alexnet , GoogLeNet , ResNet (ResNet18, ResNet50, ResNet101), Inception-V3 , Inception-ResNet-V2 , SqueezeNet , and MobileNet-V2 were trained to classify eardrum and external auditory canal features into six categories of ear diseases, covering most ear diseases (Normal, Attic retraction, Tympanic perforation, Otitis externa±myringitis, Tumor). The best two among nine models were selected by evaluating the performance of each model in terms of accuracy and calculation time. An ensemble classifier is generated that combines classifiers' outputs from the best two models. Each classifier scores the probability of an input image to be one of six classes and the maximal score among all classes is chosen as a predicted label. Overall, the ensemble system was able to achieve an average of 93·73% diagnostic accuracy. [2]

2.4.5 Applying an ensemble convolutional neural network with Savitzky–Golay filter to construct a phonocardiogram prediction model

Year & Publication: 2019 & Applied Soft Computing

Technique: The phonocardiograms come from seven different research institutions in different clinical or non-clinical environments is used in this study. The sampling rate of each heart sound was consistent at 2000 Hz and each phonocardiogram was an uncompressed wave format file. The method in this paper applies a filter to deal with the noise of heart sounds. Therefore, the performance of phonocardiogram classification model is improved. This study

uses three features (Spectrogram, Mel Spectrogram, and MFCCs) as input for the model to build the classifier for phonocardiogram classification. First, a Savitzky-Golay filter is used to denoise each heart sound, and then the filtered and unfiltered time domain signals are transformed to the frequency domain by short-time Fourier transform. The ensemble model is built through two ensemble strategies, and the prediction results of multiple models (CNNs) are combined by a majority voting method. Finally, the final prediction result is output by the ensemble model to determine whether the phonocardiograms are normal or abnormal. The average sensitivity was 91.73%, the average specificity was 87.9%, and the average MAcc was 89.81%. [22]

2.5 DRAWBACKS OF EXISTING SYSTEM

- One of the major drawbacks of the existing systems include lack of deployment validation. The existing models do not show credibility on their practical application.
- Training deep neural networks with limited data may lead to over-fitting and negatively impact the model's ability to generalize on new test data.
- There exist no appreciable research in the field of on-device detection of Alzheimer's disease. Owing to the high computation requirement of Deep Learning models, sensitive health data is sent to cloud servers. This induces a two-way round trip time and thus the response time is relatively high. Cloud servers also open up the risk of privacy breach of healthcare data which is meant to be confidential.

- Furthermore, the dataset chosen has a major class imbalance problem. Among the four classes, one of the class has only 64 images, whereas other classes have hundreds of images.

2.6 PROBLEM STATEMENT

- To solve the class imbalance problem by applying Generative Adversarial Networks (GAN) and other oversampling techniques.
- To increase the efficiency and sensitivity of detection by using ensemble networks to classify MRI images.
- To deploy on low resource microprocessors and showcase the ability of our deep learning model.
- To observe and report the total inference time for our model to classify test subject's MRI image as dementia or not.

2.7 SUMMARY

Various literature were studied and surveyed to understand different implementation strategies. Based on the literature studied, the drawbacks of the existing systems were understood and thus the motivation of the project was derived. The system requirements of the project are stated in the next chapter.

CHAPTER 3

SYSTEM REQUIREMENTS

This chapter gives details on the hardware and software requirements for this project.

3.1 HARDWARE REQUIREMENTS

- **Processor**: Any processor with four or more physical cores.
- **Memory**: 16GB or more.
- **Storage**: Dependent on the dataset and model size. Preferably 8GB or more.
- **Edge-device**: Raspberry Pi 4 Model B with 2GB RAM.

3.2 SOFTWARE REQUIREMENTS

- **Development Environment**: Anaconda and Kaggle
- **Computational Environment**: Jupyter Notebook
- **Programming Language**: Python3
- **ML Framework**: TensorFlow & Keras

3.3 OPERATING SYSTEM

- **Development OS**: Windows 10/11 64-bit or Ubuntu 20.04 64bit
- **Deployment OS**: Raspbian OS 64-bit

3.4 SUMMARY

The fundamental hardware, software, and OS requirements have been specified in this chapter. The next chapter covers the design of the project flow and explains each module in detail.

CHAPTER 4

DESIGN

This chapter discusses the Synthetic Minority Oversampling Techinque (SMOTE), Generative Adversarial Networks (GAN), and Convolutional Neural Network (CNN) in detail. This chapter also explains the required modules.

4.1 INTRODUCTION

The project aims at improving the accuracy of the prediction as well as the parameters of efficiency, such as inference time and hardware power requirement. This is because good accuracy is desirable, specifically sensitivity so that there are as few false negatives as possible since a false negative is significantly worse than a false positive when it comes to diseases like Alzheimer's. We also want to increase the efficiency parameters so that this can be used on edge devices without a connection to the cloud.

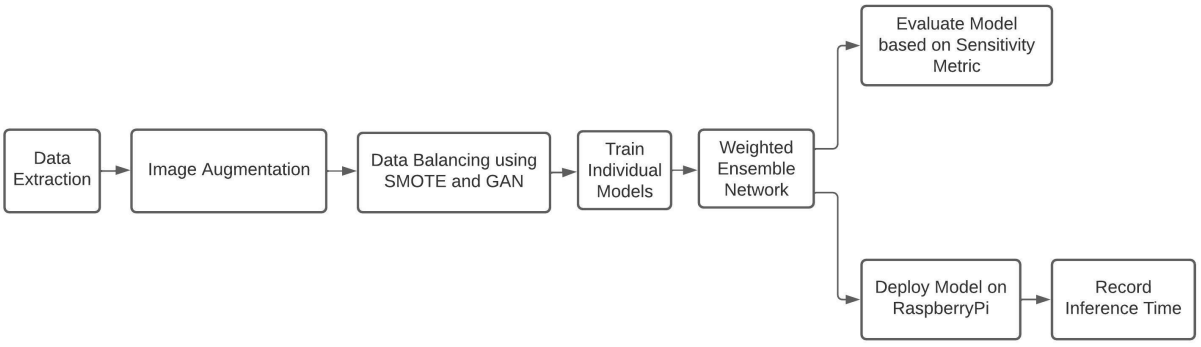

FIGURE 4.1: Proposed System Architecture

Obtaining a sufficient and consistent set of samples is one of the most important tasks as the datasets are medical, so their balance and availability are inconsistent at best. As depicted in the architecture diagram, the data will be preprocessed. This includes normalizing the data, creating class and label combinations for the various types of dementia present. We also do various sanity checks to see whether the dataset is free of errors and is balanced or not. Next, during the image augmentation process, images are modified to create a varied dataset to prevent overfitting. If the data is imbalanced, GANs for creating artificial MRI scans and SMOTE for oversampling will be used. A GAN creates artificial images by training two networks, a generator and a discriminator. The generator tries to create realistic images, and the discriminator tries to discriminate real images from artificial ones. By synthesizing images for the minority class, the model can broaden its decision-making for the minority class. The balanced dataset is now fed to multiple models, with appropriate augmentations performed. The models are trained with the primary loss metric being sensitivity. Once the models are trained, they arrive at a decision together using an ensemble network, which will be trained separately. The models will not be trained at this step, and they will act as a black box. Once the entire ensemble is trained, the complete model will be deployed onto edge devices like the Raspberry Pi and the inference time will be recorded. Fig. 4.1 encapsulates the proposed system into an illustration.

4.2 MODULE SPLIT-UP

- Extraction of Image Data
- Image Augmentation

- Train Individual Classifier Models
- Weighted Ensemble Network
- Model Evaluation
- Deploy Model on RaspberryPi
- Record Inference Time

4.3 MODULE DESCRIPTION

4.3.1 Data Extraction

The dataset is downloaded from **Kaggle** using the Kaggle API [4]. The data comprises 6400 Magnetic Resonance Imaging (MRI) scans of various patients and is provided as a multi-class classification task.

Class	**Number of Samples**
NonDemented	3200
VeryMildDemented	2240
MildDemented	896
ModerateDemented	64
Total	**6400**

TABLE 4.1: Table consisting number of samples for each class

The number of samples for Mild Demented and Moderate Demented classes are less in comparison with Non-Demented and Very Mild Demented classes. This

calls for a need to oversample the entire data to make meaningful predictions. Fig. 4.2 depicts sample images of all 4 classes. Refer to Table. 4.1 for class distribution.

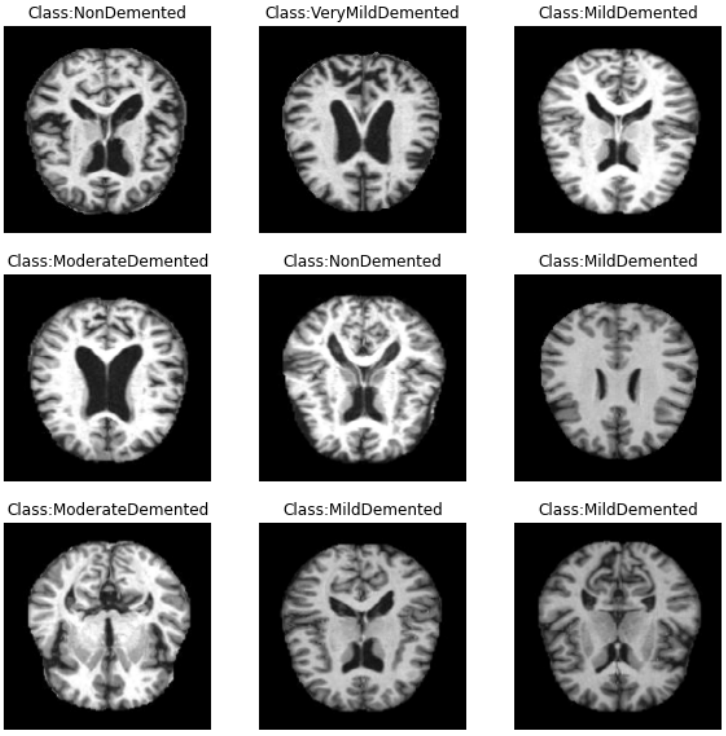

FIGURE 4.2: Images of all classes

4.3.2 Image Augmentation

Image Data Augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images. [15]

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the *ImageDataGenerator* class. Deep

```
In [4]: #Performing Image Augmentation to have more data samples

        ZOOM = [.99, 1.01]
        BRIGHT_RANGE = [0.8, 1.2]
        HORZ_FLIP = True
        FILL_MODE = "constant"
        DATA_FORMAT = "channels_last"

        work_dr = IDG(rescale = 1./255,
                      brightness_range=BRIGHT_RANGE,
                      zoom_range=ZOOM,
                      data_format=DATA_FORMAT,
                      fill_mode=FILL_MODE,
                      horizontal_flip=HORZ_FLIP)
```

FIGURE 4.3: Image Data Augmentation

networks need large amount of training data to achieve good performance. This has been implemented as seen in Fig. 4.3.

To build a powerful image classifier using very little training data, image augmentation is usually required to boost the performance of deep networks. Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear, flips, etc as given in Fig. 4.4.

Data augmentation is a strategy that enables practitioners to significantly increase the diversity of data available for training models, without actually collecting new data. Data augmentation is another way we can reduce overfitting on models, where we increase the amount of training data using information only in our training data. As can be seen in Fig. 4.4, input images are augmented through a series of image manipulation functions.

4.3.3 Data Balancing using SMOTE and GAN

Data balancing is a very important part of data augmentation. The data fed to the model should be balanced to make sure the model makes an accurate prediction for all the classes. If a class has noticeably fewer data points compared to the

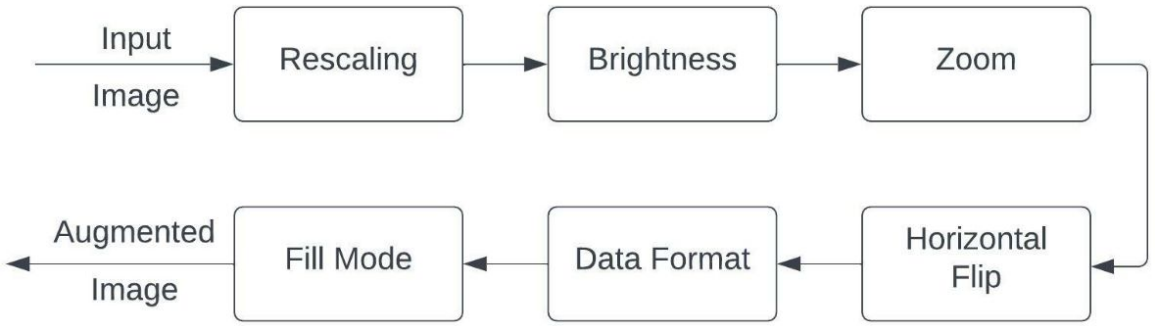

FIGURE 4.4: Image Data Augmentation Flow Chart

others, it can affect the model's performance on that class, since it does not have the same knowledge about the minority class to make a competent decision. This can be solved by multiple ways. But here we focus on two ways which best fit the problem statement. Fig. 4.5 clearly depicts the flow of this process.

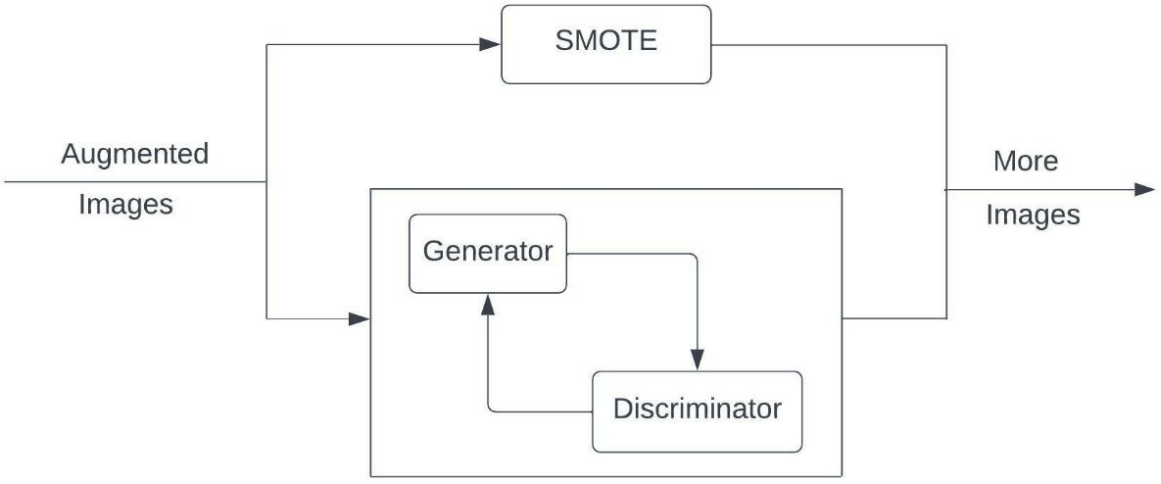

FIGURE 4.5: Data Balancing using SMOTE and GAN Flow Chart

Synthetic Minority Oversampling Technique or **SMOTE** is a method which involves oversampling the minority class by synthesizing new examples from existing examples [3]. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space, and

drawing a new sample at a point along that line. This can significantly increase the number of examples in the minority class. We can reduce the rate of overfitting by also under sampling the majority class to match the over sampled minority class.

In a **Generative Adversarial Network** or **GAN**, two neural networks contest with each other [6]. Given a training set, this technique learns to generate new data with the same statistics as the training set. The first network called the generator tries to generate realistic looking images as possible from random noise. The other network called the discriminator tries to differentiate artificial images produced by the generator from real ones. By training these models against each other, the generator produces more and more realistic looking images and the discriminator becomes better at spotting out artificially produced images. After significant training cycles, the generator now can be used to produce artificial images which look very similar to the original ones. This can be used on the minority class to produce artificial images to increase the training examples.

4.3.4 Train individual models

This project uses individual models namely InceptionV3, DenseNet-121, and AlexNet as independent models that will be used for the **Weighted Ensemble Network** discussed in Section 5.8. The model code, model summary, and model execution along with the results are described in length under Section 5.5, 5.6, 5.7, and 5.8.

4.3.5 Weighted Ensemble Network

This project will be using an ensemble network to arrive at a decision based on the outputs of the models. To make this fully self-contained from end to end, the models along with the different types of inputs will be included as part of the network. Each model is trained separately. They only act as a black box here and are not trained during this step. There are three layers in this network. The first one contains the input images. This input layer is connected to the corresponding models in the model layer. In this layer, each model acts as a black box and gives output for the input given without being trained. Each of these outputs is connected to the output node, which takes a weighted average of the inputs it receives to arrive at a decision. The weights used for the weighted average will be optimized by training the ensemble network as a whole. Fig. 4.6 depicts a tentative flow diagram of the weighted ensemble network.

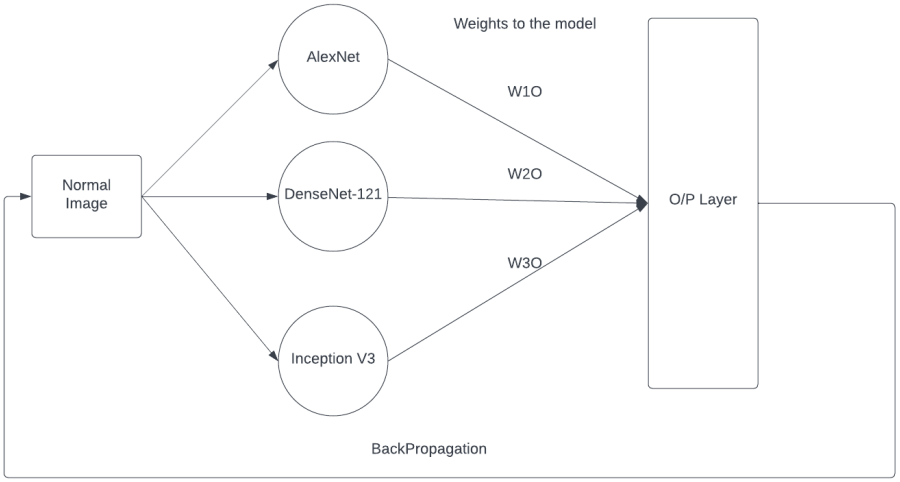

FIGURE 4.6: Weighted Ensemble Network

4.3.6 Model Evaluation

The individual models are evaluated based on the metrics loss, accuracy, AUC Score, F1-score. The definitions of the same are given in equations 4.1, 4.2, 4.3, 4.4, and 4.5.

$$Loss = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{4.1}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.3}$$

$$Sensitivity = Recall = \frac{TP}{TP + FN} \tag{4.4}$$

$$Specificity = \frac{TN}{FP + TN} \tag{4.5}$$

TP = Dementia events classified as dementia events FP = Non-dementia events classified as dementia events TN = Non-dementia events classified as non-dementia events FN = Dementia events classified as non-dementia events AUC ROC is the area under the Sensitivity and (1 - Specificity) curve.

4.3.7 Model Deployment and Inference Time Observation

The **Weighted Ensemble Network** is to be offloaded onto a microcomputer namely **Raspberry Pi 4 Model B**, as can be inferred from Fig. 4.7. An illustration of the Raspberry Pi 4 used is shown in Fig. 5.32. The network is expected to be fully trained and cognizant of the general characteristics pertaining

SoC	64-bit AArch quad-core Cortex-A72
RAM	2GB
Operating Power	5V/3A DC power unit
Clock Speed	1.5GHz

TABLE 4.2: Edge-device specifications

to Alzheimer's Disease. The specifications of the edge-device (microcomputer) are given in the Table 4.2 below.

A Python script will be written to calculate mean inference time for the end-to-end classification after running for a couple hundred cycles. InceptionV3, DenseNet-121, and AlexNet were trained and tested on Kaggle runtime and on personal resource (Nvidia GTX 1650, 4GB VRAM, Ubuntu 20.04). GAN was trained and tested on desktop computer (Nvidia RTX 3060 Ti, 10GB VRAM, Arch Linux).

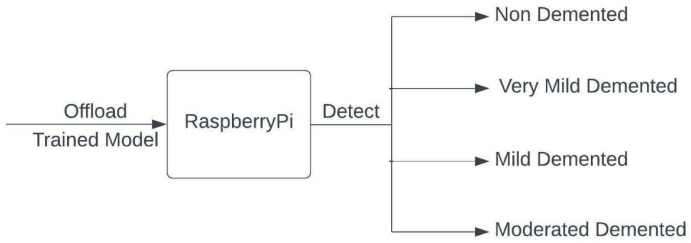

FIGURE 4.7: Deployment Flow Chart

4.4 SUMMARY

This chapter has elaborated the overall description of the project and its required algorithms. The demonstration is provided in the next chapter.

CHAPTER 5

IMPLEMENTATION

The sections below in this chapter explain in detail the implementation phase of the project.

5.1 ANACONDA NAVIGATOR

Anaconda Navigator is a desktop graphical user interface (GUI) in Anaconda distribution that allows to launch applications. The Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository and eliminates the need of command-line commands. Also the Navigator helps to the find the required packages, install them in an environment, run the packages, and update them. Fig. 5.1 shows the dialog box for installation of Anaconda Navigator.

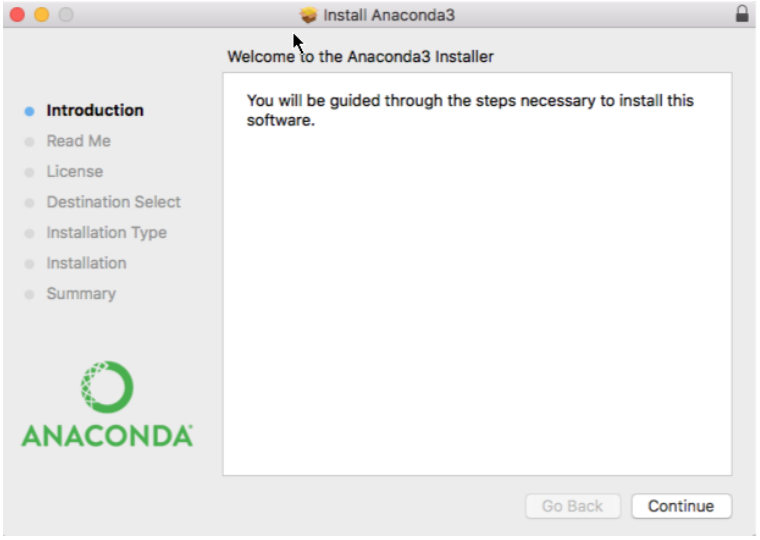

FIGURE 5.1: Anaconda Navigator Installation

5.2 JUPYTER NOTEBOOK

Jupyter Notebook is a default application launched by the Anaconda Navigator. It is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala. The implementation of this project uses Python with version 3.9.2. Figure 5.2 shows how to launch Jupyter Notebook from Anaconda Navigator.

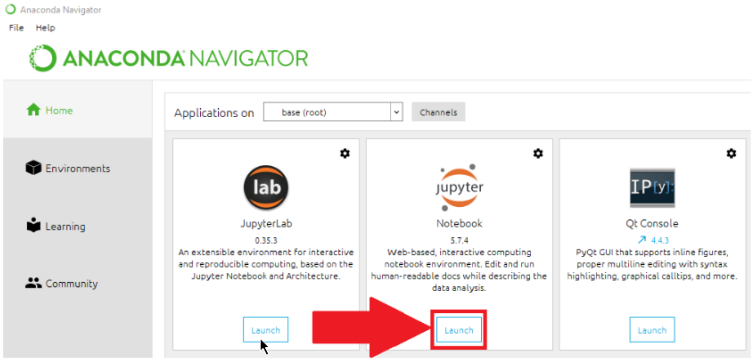

FIGURE 5.2: Launching Jupyter Notebook

5.3 TENSORFLOW AND KERAS

TensorFlow is an end-to-end open source platform for Machine Learning (ML). It has a comprehensive, flexible ecosystem of tools, libraries and community resources that helps to build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging. TensorFlow 2.9.0 was used for building the network. Fig. 5.3 shows the installation of TensorFlow with Pip Python.

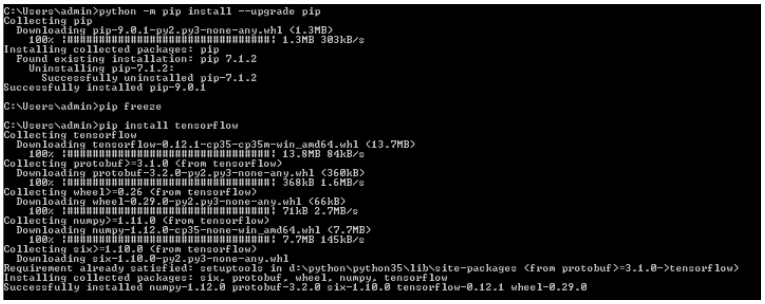

FIGURE 5.3: Installing Tensorflow

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It was developed with a focus on enabling fast experimentation. Beyond ease of learning and ease of model building, Keras offers the advantages of broad adoption, support for a wide range of production deployment options. Fig. 5.4 shows the installation of Keras with TensorFlow backend.

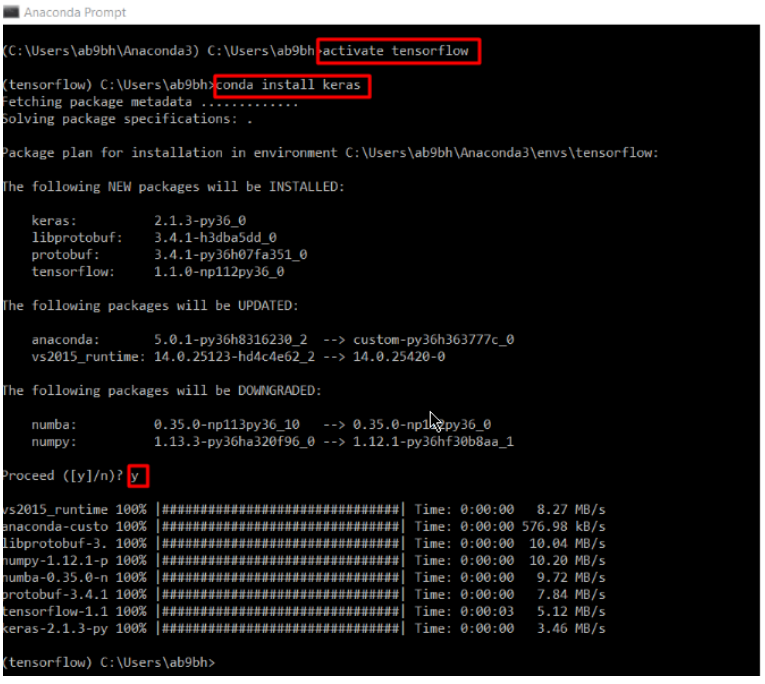

FIGURE 5.4: Installing Keras

5.4 GENERATIVE ADVERSARIAL NETWORKS

A Generative Adversarial Network model was trained on down sampled 56×56 MRI scan images to produce artificial images of size 56×56 over 50 epochs with a similar sized generator and discriminator with the same learning rate. Fig. 5.5, 5.6 depict the generator and discriminator code, and Fig. 5.7, 5.8 shows a visualization of their architectures.

5.4.1 GAN Code

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 28, 28, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (3, 3), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 56, 56, 1)

    return model
```

FIGURE 5.5: Generator Model Code

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                    input_shape=[56, 56, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

FIGURE 5.6: Discriminator Model Code

5.4.2 GAN Architecture

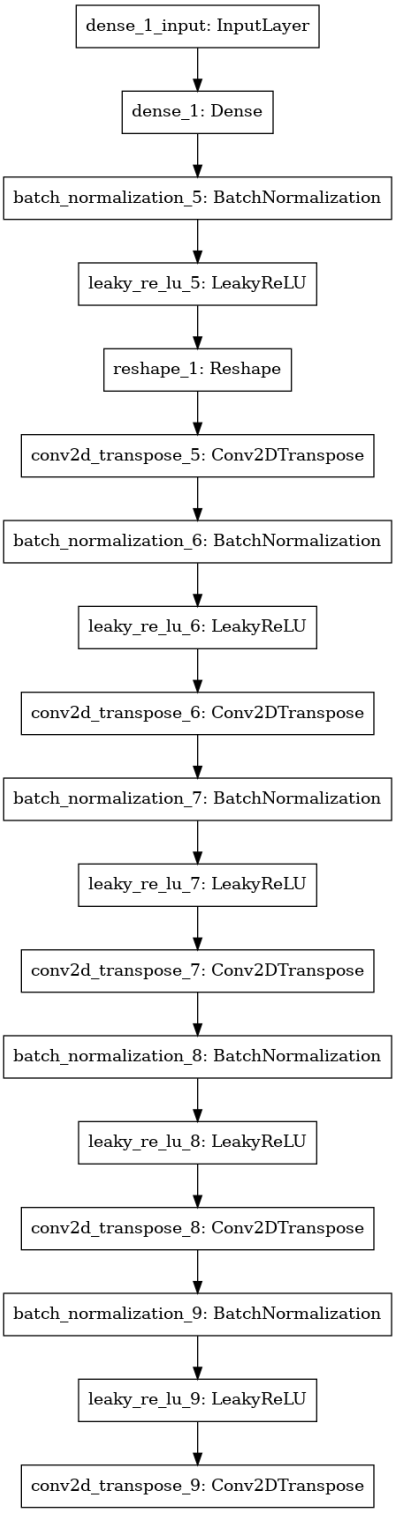

FIGURE 5.7: Generator Model Architecture

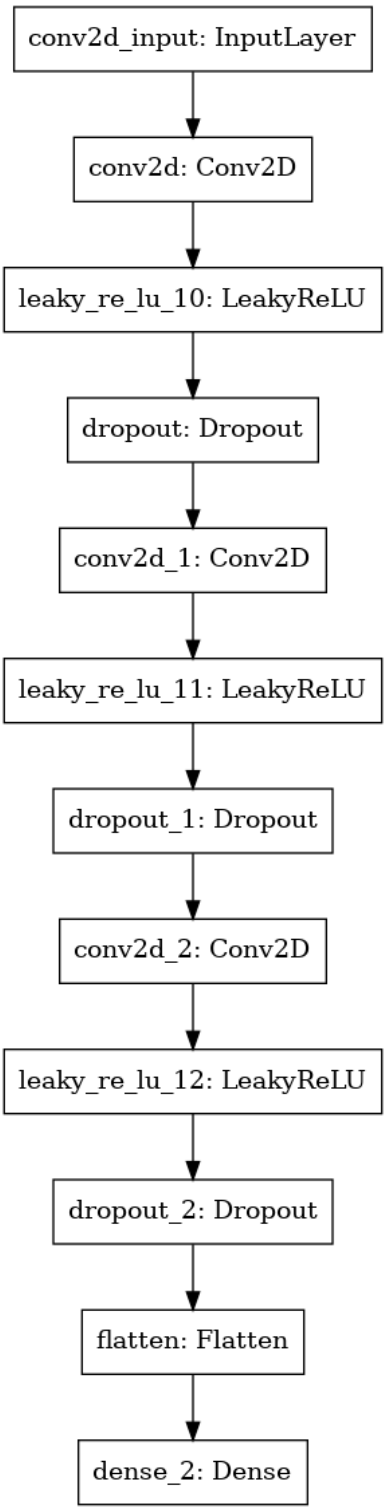

FIGURE 5.8: Discriminator Model Architecture

5.4.3 GAN Execution and Results

Fig. 5.9, 5.10 are the generated images by the generator at epochs 1 and 50 respectively.

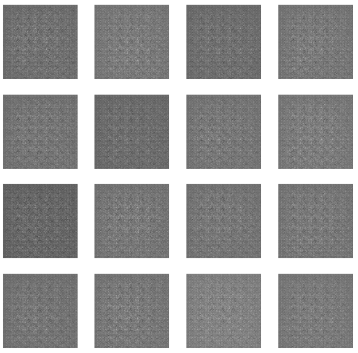

FIGURE 5.9: Generated Images at epoch 1

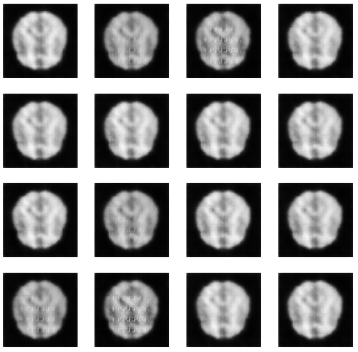

FIGURE 5.10: Generated Images at epoch 50

5.4.4 GAN Failure

Even after trying a custom GAN with 23 million parameters and producing a high resolution synthetic image, the resulting synthetic images did not have

satisfactory granularity and detail required to make precise predictions. The generator is dominated by the discriminator because of the complexity of the image, and thus it is never able to produce finer details. Since this would negatively affect the performance of the neural network, we have decided to not proceed further with GAN as an effective method for image balancing and instead focus our effort on SMOTE.

5.5 INCEPTIONV3

The Inception V3 is a deep learning model based on Convolutional Neural Networks, which is used for image classification [20]. The inception V3 is a superior version of the basic model Inception V1 which was introduced as GoogLeNet in 2014.

The Inception V3 is just the advanced and optimized version of the Inception V1 model. The Inception V3 model used several techniques for optimizing the network for better model adaptation.

- It has higher efficiency.
- It has a deeper network compared to the Inception V1 and V2 models, but its speed is not compromised.
- It is computationally less expensive.
- It uses auxiliary classifiers as regularizers.

5.5.1 InceptionV3 Code

In [12]:
```
custom_inception_model = Sequential([
        inception_model,
        Dropout(0.5),
        GlobalAveragePooling2D(),
        Flatten(),
        BatchNormalization(),
        Dense(512, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(256, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(128, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.5),
        BatchNormalization(),
        Dense(4, activation='softmax')
    ], name = "inception_cnn_model")
```

In [13]:
```
#Defining a custom callback function to stop training our model when accuracy goes above 99%

class MyCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('acc') > 0.99:
            print("\nReached accuracy threshold! Terminating training.")
            self.model.stop_training = True

my_callback = MyCallback()

#ReduceLROnPlateau to stabilize the training process of the model
rop_callback = ReduceLROnPlateau(monitor="val_loss", patience=3)
```

In [14]:
```
METRICS = [tf.keras.metrics.CategoricalAccuracy(name='acc'),
           tf.keras.metrics.AUC(name='auc'),
           tfa.metrics.F1Score(num_classes=4),
           tf.keras.metrics.Recall(name='recall')]

CALLBACKS = [my_callback, rop_callback]

custom_inception_model.compile(optimizer='rmsprop',
                               loss=tf.losses.CategoricalCrossentropy(),
                               metrics=METRICS)

custom_inception_model.summary()
```

FIGURE 5.11: InceptionV3 Model Code

Fig. 5.11 depicts the code written for InceptionV3 model using TensorFlow.

5.5.2 InceptionV3 Summary

```
Model: "inception_cnn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 4, 4, 2048)        21802784
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 2048)        0
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
batch_normalization_94 (Batc (None, 2048)              8192
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
batch_normalization_95 (Batc (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_96 (Batc (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_97 (Batc (None, 128)               512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
batch_normalization_98 (Batc (None, 64)                256
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 260
=================================================================
Total params: 23,036,644
Trainable params: 1,227,844
Non-trainable params: 21,808,800
_________________________________________________________________
```

FIGURE 5.12: InceptionV3 Summary

Fig. 5.12 illustrates the summary of the custom InceptionV3 model. This custom InceptionV3 model has its initial layer transferred from Google's InceptionV3. The layers are frozen, while the last layer is made trainable. It is connected to a set of Batch Normalization, Dropout, and Dense layers making the overall parameters in the order of 23 million.

5.5.3 InceptionV3 Execution and Results

```
Epoch 1/100
2022-03-22 08:14:09.145740: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
255/256 [============================>.] - ETA: 0s - loss: 1.3794 - acc: 0.4020 - auc: 0.6720 - f1_score: 0.3967 -
recall: 0.2516
2022-03-22 08:14:25.277677: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 761266176 exceeds
10% of free system memory.
2022-03-22 08:14:26.050702: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 761266176 exceeds
10% of free system memory.
256/256 [==============================] - 28s 63ms/step - loss: 1.3780 - acc: 0.4027 - auc: 0.6726 - f1_score: 0.
3975 - recall: 0.2518 - val_loss: 0.8448 - val_acc: 0.6045 - val_auc: 0.8688 - val_f1_score: 0.5886 - val_recall:
0.3843
Epoch 2/100
256/256 [==============================] - 13s 52ms/step - loss: 0.9289 - acc: 0.5648 - auc: 0.8399 - f1_score: 0.
5559 - recall: 0.4032 - val_loss: 0.7049 - val_acc: 0.6494 - val_auc: 0.9041 - val_f1_score: 0.5935 - val_recall:
0.5444
Epoch 3/100
256/256 [==============================] - 13s 52ms/step - loss: 0.7961 - acc: 0.6328 - auc: 0.8836 - f1_score: 0.
6238 - recall: 0.4747 - val_loss: 0.6277 - val_acc: 0.6997 - val_auc: 0.9245 - val_f1_score: 0.6868 - val_recall:
0.5527
Epoch 4/100
256/256 [==============================] - 13s 51ms/step - loss: 0.7400 - acc: 0.6649 - auc: 0.9005 - f1_score: 0.
6610 - recall: 0.5159 - val_loss: 0.5997 - val_acc: 0.7217 - val_auc: 0.9322 - val_f1_score: 0.7180 - val_recall:
0.5479
Epoch 5/100
256/256 [==============================] - 13s 51ms/step - loss: 0.6975 - acc: 0.6831 - auc: 0.9116 - f1_score: 0.
6817 - recall: 0.5419 - val_loss: 0.5761 - val_acc: 0.7354 - val_auc: 0.9370 - val_f1_score: 0.7250 - val_recall:
0.6025
Epoch 6/100
256/256 [==============================] - 13s 52ms/step - loss: 0.6587 - acc: 0.7091 - auc: 0.9216 - f1_score: 0.
7066 - recall: 0.6027 - val_loss: 0.5394 - val_acc: 0.7568 - val_auc: 0.9470 - val_f1_score: 0.7581 - val_recall:
0.6895
Epoch 7/100
256/256 [==============================] - 13s 51ms/step - loss: 0.6371 - acc: 0.7211 - auc: 0.9269 - f1_score: 0.
7204 - recall: 0.6340 - val_loss: 0.5178 - val_acc: 0.7686 - val_auc: 0.9507 - val_f1_score: 0.7650 - val_recall:
0.6875
Epoch 8/100
256/256 [==============================] - 13s 52ms/step - loss: 0.6028 - acc: 0.7432 - auc: 0.9353 - f1_score: 0.
7435 - recall: 0.6726 - val_loss: 0.4973 - val_acc: 0.7852 - val_auc: 0.9550 - val_f1_score: 0.7808 - val_recall:
0.7144
Epoch 9/100
256/256 [==============================] - 13s 50ms/step - loss: 0.5835 - acc: 0.7513 - auc: 0.9391 - f1_score: 0.
7508 - recall: 0.6912 - val_loss: 0.4586 - val_acc: 0.8110 - val_auc: 0.9627 - val_f1_score: 0.8089 - val_recall:
0.7578
Epoch 10/100
256/256 [==============================] - 13s 50ms/step - loss: 0.5450 - acc: 0.7793 - auc: 0.9477 - f1_score: 0.
7782 - recall: 0.7308 - val_loss: 0.4466 - val_acc: 0.8096 - val_auc: 0.9636 - val_f1_score: 0.8090 - val_recall:
0.7822
```

FIGURE 5.13: First 10 epochs of InceptionV3

```
Epoch 90/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1980 - acc: 0.9323 - auc: 0.9917 - f1_score: 0.
9321 - recall: 0.9281 - val_loss: 0.2668 - val_acc: 0.9009 - val_auc: 0.9875 - val_f1_score: 0.8999 - val_recall:
0.8975
Epoch 91/100
256/256 [==============================] - 13s 52ms/step - loss: 0.2021 - acc: 0.9315 - auc: 0.9916 - f1_score: 0.
9315 - recall: 0.9255 - val_loss: 0.2679 - val_acc: 0.9004 - val_auc: 0.9871 - val_f1_score: 0.8994 - val_recall:
0.8965
Epoch 92/100
256/256 [==============================] - 13s 51ms/step - loss: 0.2084 - acc: 0.9318 - auc: 0.9905 - f1_score: 0.
9317 - recall: 0.9270 - val_loss: 0.2671 - val_acc: 0.9004 - val_auc: 0.9871 - val_f1_score: 0.8994 - val_recall:
0.8979
Epoch 93/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1953 - acc: 0.9346 - auc: 0.9920 - f1_score: 0.
9344 - recall: 0.9286 - val_loss: 0.2671 - val_acc: 0.9019 - val_auc: 0.9870 - val_f1_score: 0.9010 - val_recall:
0.8999
Epoch 94/100
256/256 [==============================] - 13s 52ms/step - loss: 0.2076 - acc: 0.9318 - auc: 0.9908 - f1_score: 0.
9316 - recall: 0.9254 - val_loss: 0.2689 - val_acc: 0.8999 - val_auc: 0.9868 - val_f1_score: 0.8988 - val_recall:
0.8960
Epoch 95/100
256/256 [==============================] - 13s 51ms/step - loss: 0.2029 - acc: 0.9315 - auc: 0.9913 - f1_score: 0.
9315 - recall: 0.9266 - val_loss: 0.2693 - val_acc: 0.8999 - val_auc: 0.9874 - val_f1_score: 0.8988 - val_recall:
0.8965
Epoch 96/100
256/256 [==============================] - 13s 52ms/step - loss: 0.2011 - acc: 0.9325 - auc: 0.9914 - f1_score: 0.
9324 - recall: 0.9276 - val_loss: 0.2678 - val_acc: 0.8999 - val_auc: 0.9868 - val_f1_score: 0.8989 - val_recall:
0.8975
Epoch 97/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1918 - acc: 0.9327 - auc: 0.9922 - f1_score: 0.
9326 - recall: 0.9266 - val_loss: 0.2674 - val_acc: 0.9009 - val_auc: 0.9868 - val_f1_score: 0.8998 - val_recall:
0.8989
Epoch 98/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1824 - acc: 0.9386 - auc: 0.9933 - f1_score: 0.
9385 - recall: 0.9351 - val_loss: 0.2668 - val_acc: 0.9014 - val_auc: 0.9873 - val_f1_score: 0.9005 - val_recall:
0.8989
Epoch 99/100
256/256 [==============================] - 13s 51ms/step - loss: 0.2056 - acc: 0.9299 - auc: 0.9915 - f1_score: 0.
9298 - recall: 0.9244 - val_loss: 0.2677 - val_acc: 0.9014 - val_auc: 0.9869 - val_f1_score: 0.9003 - val_recall:
0.8975
Epoch 100/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1927 - acc: 0.9355 - auc: 0.9922 - f1_score: 0.
9354 - recall: 0.9299 - val_loss: 0.2656 - val_acc: 0.8989 - val_auc: 0.9876 - val_f1_score: 0.8980 - val_recall:
0.8965
CPU times: user 15min 52s, sys: 1min 2s, total: 16min 55s
Wall time: 22min 32s
```

FIGURE 5.14: Last 10 epochs of InceptionV3

Fig. 5.13 and Fig. 5.14 represent the first and last 10 epochs of InceptionV3 training and validation respectively.

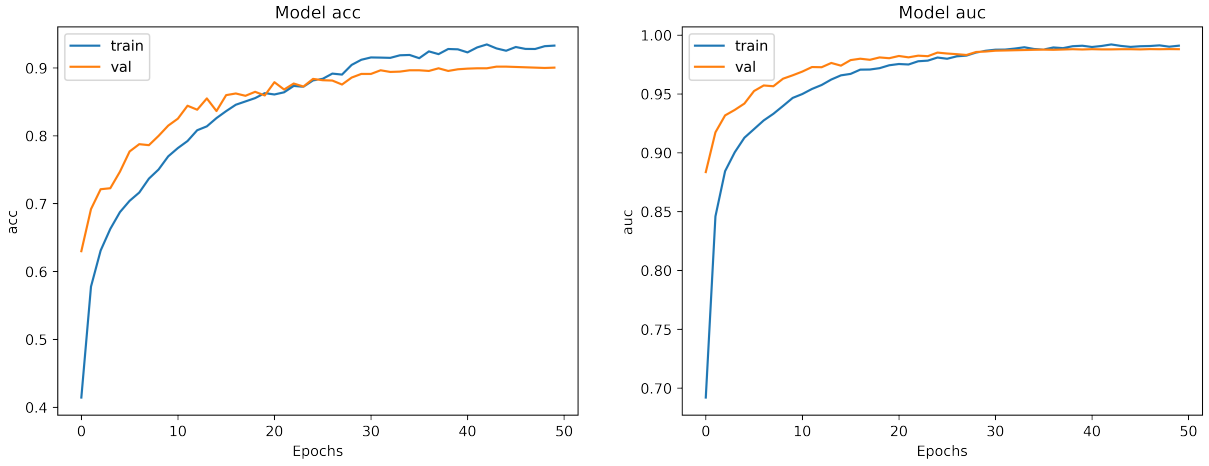

FIGURE 5.15: InceptionV3 accuracy and auc plot

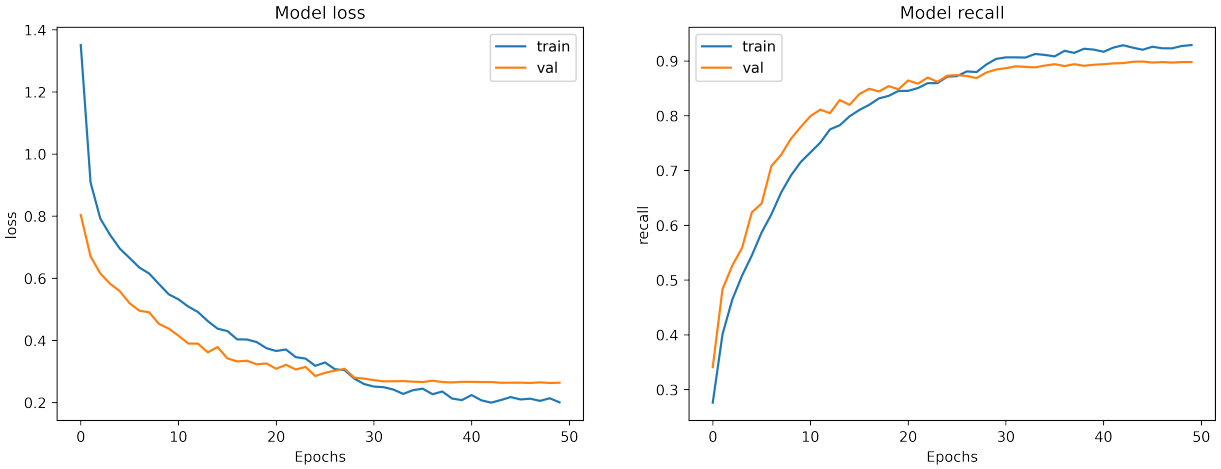

FIGURE 5.16: InceptionV3 loss and recall plot

Fig. 5.15 and Fig. 5.16 shows that the train and val accuracy increase together with little fluctuation in validation accuracy until around epoch 25, after which train accuracy surpasses validation accuracy. The same can be said for model recall. The AUCROC score for both train and validation data increases until epoch 25. However, they both stabilize at a similar AUCROC score. Train and validation loss fall in a stable manner until epoch 25, after which they both stabilize with slight fluctuations in train loss.

5.6 DENSENET

DenseNet is one of the new discoveries in neural networks for visual object recognition [8]. DenseNet is quite similar to ResNet with some fundamental differences. ResNet uses an additive method (+) that merges the previous layer (identity) with the future layer, whereas DenseNet concatenates (.) the output of the previous layer with the future layer.

5.6.1 DenseNet Code

```
model = Sequential([
        densenet_model,
        Dropout(0.5),
        GlobalAveragePooling2D(),
        Flatten(),
        BatchNormalization(),
        Dense(512, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(256, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(128, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.5),
        BatchNormalization(),
        Dense(4, activation='softmax')
    ], name = "DenseNet121")
```

```
#Defining a custom callback function to stop training our model when accuracy goes above 99%

class MyCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('acc') > 0.99:
            print("\nReached accuracy threshold! Terminating training.")
            self.model.stop_training = True

my_callback = MyCallback()

#ReduceLROnPlateau to stabilize the training process of the model
rop_callback = ReduceLROnPlateau(monitor="val_loss", patience=3)
```

```
METRICS = [tf.keras.metrics.CategoricalAccuracy(name='acc'),
           tf.keras.metrics.AUC(name='auc'),
           tfa.metrics.F1Score(num_classes=4),
           tf.keras.metrics.Recall(name='recall')]

CALLBACKS = [my_callback, rop_callback]

model.compile(optimizer='rmsprop',
                            loss=tf.losses.CategoricalCrossentropy(),
                            metrics=METRICS)

model.summary()
```

FIGURE 5.17: DenseNet Model Code

Fig. 5.17 depicts the code written for DenseNet121 model using TensorFlow.

5.6.2 DenseNet Summary

```
Model: "DenseNet121"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Functional)     (None, 5, 5, 1024)        7037504
_________________________________________________________________
dropout (Dropout)            (None, 5, 5, 1024)        0
_________________________________________________________________
global_average_pooling2d (Gl (None, 1024)              0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 1024)              4096
_________________________________________________________________
dense (Dense)                (None, 512)               524800
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
batch_normalization_4 (Batch (None, 64)                256
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 260
=================================================================
Total params: 7,742,980
Trainable params: 701,508
Non-trainable params: 7,041,472
```

FIGURE 5.18: DenseNet Summary

Fig. 5.18 shows the summary for the custom DenseNet-121 model written using TensorFlow. The last few layers of the DenseNet are frozen, and is followed by a

series of batch normalization, dropout and dense layers that brings the total number of parameters to shy of 8 million.

5.6.3 DenseNet Execution and Results

```
Epoch 1/50
2022-04-11 15:58:46.438281: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
256/256 [==============================] - ETA: 0s - loss: 1.3180 - acc: 0.4304 - auc: 0.7034 - f1_score: 0.4278 - recall:
0.2927
2022-04-11 15:59:03.406631: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 761266176 exceeds 10% of fre
e system memory.
2022-04-11 15:59:04.242315: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of 761266176 exceeds 10% of fre
e system memory.
256/256 [==============================] - 31s 69ms/step - loss: 1.3180 - acc: 0.4304 - auc: 0.7034 - f1_score: 0.4278 - rec
all: 0.2927 - val_loss: 0.7683 - val_acc: 0.6284 - val_auc: 0.8883 - val_f1_score: 0.6083 - val_recall: 0.4668
Epoch 2/50
256/256 [==============================] - 17s 65ms/step - loss: 0.8675 - acc: 0.6038 - auc: 0.8637 - f1_score: 0.5991 - rec
all: 0.4570 - val_loss: 0.6334 - val_acc: 0.6890 - val_auc: 0.9227 - val_f1_score: 0.6780 - val_recall: 0.5425
Epoch 3/50
256/256 [==============================] - 14s 56ms/step - loss: 0.7528 - acc: 0.6617 - auc: 0.8976 - f1_score: 0.6596 - rec
all: 0.5188 - val_loss: 0.5613 - val_acc: 0.7334 - val_auc: 0.9413 - val_f1_score: 0.7328 - val_recall: 0.5815
Epoch 4/50
256/256 [==============================] - 14s 55ms/step - loss: 0.7008 - acc: 0.6913 - auc: 0.9119 - f1_score: 0.6899 - rec
all: 0.5549 - val_loss: 0.5194 - val_acc: 0.7573 - val_auc: 0.9506 - val_f1_score: 0.7582 - val_recall: 0.6665
Epoch 5/50
256/256 [==============================] - 14s 55ms/step - loss: 0.6571 - acc: 0.7145 - auc: 0.9233 - f1_score: 0.7150 - rec
all: 0.6138 - val_loss: 0.4891 - val_acc: 0.7803 - val_auc: 0.9587 - val_f1_score: 0.7799 - val_recall: 0.6992
Epoch 6/50
256/256 [==============================] - 17s 66ms/step - loss: 0.6220 - acc: 0.7341 - auc: 0.9312 - f1_score: 0.7336 - rec
all: 0.6541 - val_loss: 0.4639 - val_acc: 0.8042 - val_auc: 0.9626 - val_f1_score: 0.8025 - val_recall: 0.7227
Epoch 7/50
256/256 [==============================] - 14s 55ms/step - loss: 0.5904 - acc: 0.7537 - auc: 0.9384 - f1_score: 0.7522 - rec
all: 0.6920 - val_loss: 0.4353 - val_acc: 0.8071 - val_auc: 0.9659 - val_f1_score: 0.8037 - val_recall: 0.7534
Epoch 8/50
256/256 [==============================] - 14s 56ms/step - loss: 0.5606 - acc: 0.7710 - auc: 0.9445 - f1_score: 0.7702 - rec
all: 0.7112 - val_loss: 0.4249 - val_acc: 0.8223 - val_auc: 0.9683 - val_f1_score: 0.8225 - val_recall: 0.7642
Epoch 9/50
256/256 [==============================] - 16s 64ms/step - loss: 0.5333 - acc: 0.7854 - auc: 0.9502 - f1_score: 0.7845 - rec
all: 0.7395 - val_loss: 0.4087 - val_acc: 0.8286 - val_auc: 0.9706 - val_f1_score: 0.8265 - val_recall: 0.7812
Epoch 10/50
256/256 [==============================] - 17s 66ms/step - loss: 0.5206 - acc: 0.7877 - auc: 0.9521 - f1_score: 0.7866 - rec
all: 0.7433 - val_loss: 0.3751 - val_acc: 0.8496 - val_auc: 0.9757 - val_f1_score: 0.8489 - val_recall: 0.8086
```

FIGURE 5.19: First 10 epochs of DenseNet

```
Epoch 41/50
256/256 [==============================] - 17s 65ms/step - loss: 0.2560 - acc: 0.9137 - auc: 0.9869 - f1_score: 0.9136 - rec
all: 0.9048 - val_loss: 0.2066 - val_acc: 0.9180 - val_auc: 0.9919 - val_f1_score: 0.9179 - val_recall: 0.9102
Epoch 42/50
256/256 [==============================] - 14s 55ms/step - loss: 0.2497 - acc: 0.9114 - auc: 0.9878 - f1_score: 0.9112 - rec
all: 0.9044 - val_loss: 0.2052 - val_acc: 0.9209 - val_auc: 0.9919 - val_f1_score: 0.9208 - val_recall: 0.9141
Epoch 43/50
256/256 [==============================] - 14s 55ms/step - loss: 0.2510 - acc: 0.9108 - auc: 0.9874 - f1_score: 0.9105 - rec
all: 0.9025 - val_loss: 0.2048 - val_acc: 0.9189 - val_auc: 0.9920 - val_f1_score: 0.9187 - val_recall: 0.9141
Epoch 44/50
256/256 [==============================] - 14s 55ms/step - loss: 0.2390 - acc: 0.9170 - auc: 0.9884 - f1_score: 0.9168 - rec
all: 0.9081 - val_loss: 0.2034 - val_acc: 0.9233 - val_auc: 0.9921 - val_f1_score: 0.9232 - val_recall: 0.9185
Epoch 45/50
256/256 [==============================] - 17s 65ms/step - loss: 0.2430 - acc: 0.9175 - auc: 0.9880 - f1_score: 0.9173 - rec
all: 0.9119 - val_loss: 0.2052 - val_acc: 0.9214 - val_auc: 0.9918 - val_f1_score: 0.9212 - val_recall: 0.9146
Epoch 46/50
256/256 [==============================] - 14s 56ms/step - loss: 0.2207 - acc: 0.9247 - auc: 0.9900 - f1_score: 0.9246 - rec
all: 0.9183 - val_loss: 0.2032 - val_acc: 0.9209 - val_auc: 0.9921 - val_f1_score: 0.9208 - val_recall: 0.9165
Epoch 47/50
256/256 [==============================] - 14s 55ms/step - loss: 0.2325 - acc: 0.9181 - auc: 0.9893 - f1_score: 0.9180 - rec
all: 0.9113 - val_loss: 0.2012 - val_acc: 0.9180 - val_auc: 0.9923 - val_f1_score: 0.9177 - val_recall: 0.9146
Epoch 48/50
256/256 [==============================] - 17s 66ms/step - loss: 0.2334 - acc: 0.9194 - auc: 0.9886 - f1_score: 0.9191 - rec
all: 0.9115 - val_loss: 0.2002 - val_acc: 0.9199 - val_auc: 0.9923 - val_f1_score: 0.9198 - val_recall: 0.9165
Epoch 49/50
256/256 [==============================] - 14s 55ms/step - loss: 0.2358 - acc: 0.9180 - auc: 0.9890 - f1_score: 0.9179 - rec
all: 0.9111 - val_loss: 0.1987 - val_acc: 0.9209 - val_auc: 0.9923 - val_f1_score: 0.9208 - val_recall: 0.9175
Epoch 50/50
256/256 [==============================] - 14s 56ms/step - loss: 0.2568 - acc: 0.9142 - auc: 0.9868 - f1_score: 0.9140 - rec
all: 0.9087 - val_loss: 0.1962 - val_acc: 0.9199 - val_auc: 0.9927 - val_f1_score: 0.9199 - val_recall: 0.9180
CPU times: user 9min 22s, sys: 39.5 s, total: 10min 1s
Wall time: 13min 34s
```

FIGURE 5.20: Last 10 epochs of DenseNet

Fig. 5.19 and Fig. 5.20 represent first and last 10 epochs of training and validating DenseNet-121 respectively.

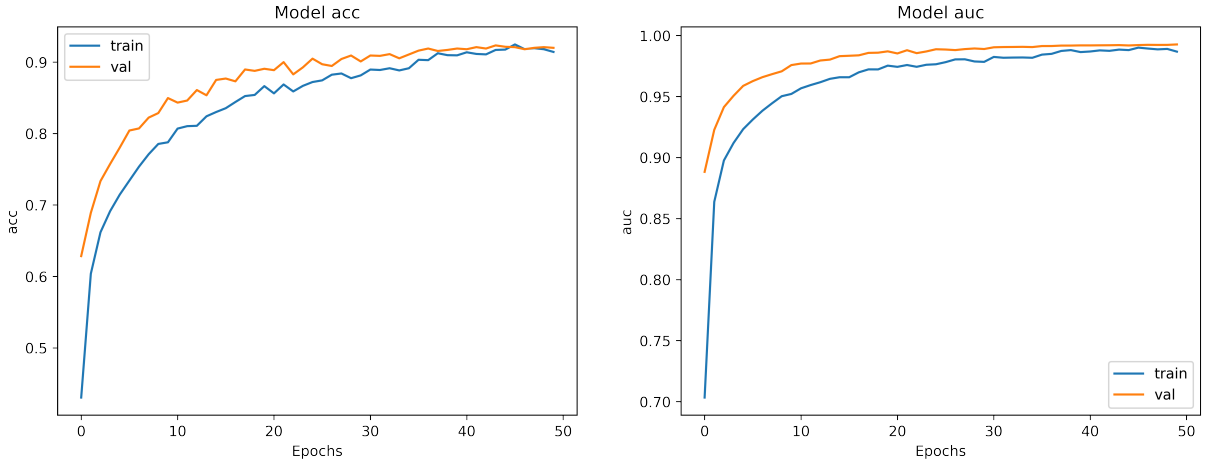

FIGURE 5.21: DenseNet121 accuracy and auc plot

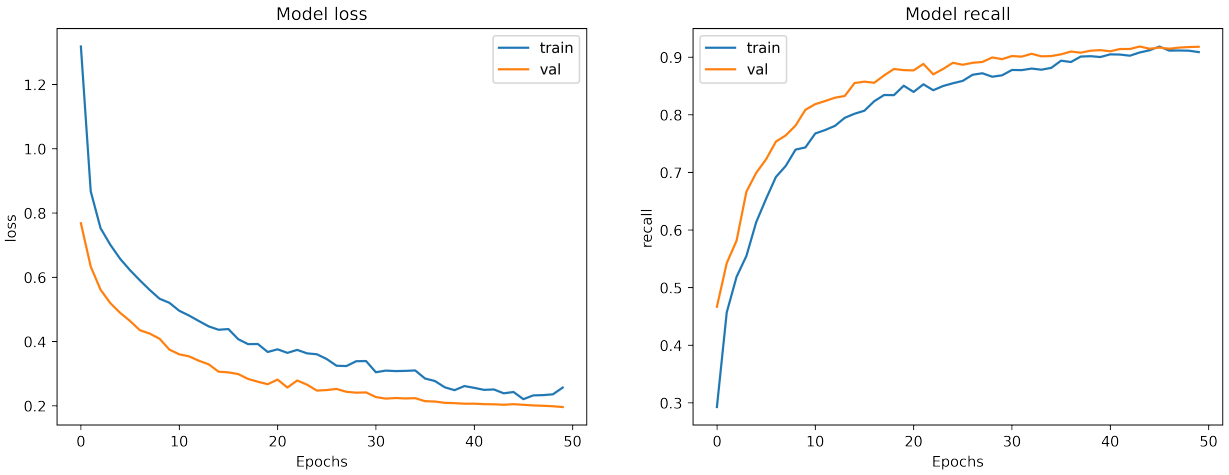

FIGURE 5.22: DenseNet121 loss and recall plot

Fig. 5.21 and Fig. 5.22 clearly show that the training and validation accuracy increase in tandem and in a consistent manner indicating a strong stable base network on top of which this learning is happening. The same can be said for the AUCROC score and recall metrics. Finally, the loss also reduces in a stable fashion and reaches a low consistent state.

5.7 ALEXNET

AlexNet was built primarily as part of the ImageNet contest in the year 2012 [10]. The primary result of the original paper was that the depth of the model was absolutely required for its high performance. This was quite expensive computationally but was made feasible due to GPUs or Graphical Processing Units, during training. The success of AlexNet is mostly attributed to its ability to leverage GPU for training and being able to train these huge numbers of parameters.

5.7.1 AlexNet Code

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(176, 176, 3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4, activation='softmax')
], name = "AlexNet")
```

```
#Defining a custom callback function to stop training our model when accuracy goes above 99%

class MyCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('acc') > 0.99:
            print("\nReached accuracy threshold! Terminating training.")
            self.model.stop_training = True

my_callback = MyCallback()

#ReduceLROnPlateau to stabilize the training process of the model
rop_callback = ReduceLROnPlateau(monitor="val_loss", patience=3)
```

```
METRICS = [tf.keras.metrics.CategoricalAccuracy(name='acc'),
           tf.keras.metrics.AUC(name='auc'),
           tfa.metrics.F1Score(num_classes=4),
           tf.keras.metrics.Recall(name='recall')]

CALLBACKS = [my_callback, rop_callback]

model.compile(optimizer='rmsprop',
                            loss=tf.losses.CategoricalCrossentropy(),
                            metrics=METRICS)

model.summary()
```

FIGURE 5.23: AlexNet Model Code

Fig. 5.23 depicts the AlexNet model code written using Tensorflow framework.

5.7.2 AlexNet Summary

```
Model: "AlexNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 42, 42, 96)        34944
_________________________________________________________________
batch_normalization_5 (Batch (None, 42, 42, 96)        384
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 20, 20, 96)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 20, 20, 256)       614656
_________________________________________________________________
batch_normalization_6 (Batch (None, 20, 20, 256)       1024
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 9, 9, 256)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 9, 9, 384)         885120
_________________________________________________________________
batch_normalization_7 (Batch (None, 9, 9, 384)         1536
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 9, 9, 384)         1327488
_________________________________________________________________
batch_normalization_8 (Batch (None, 9, 9, 384)         1536
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 9, 9, 256)         884992
_________________________________________________________________
batch_normalization_9 (Batch (None, 9, 9, 256)         1024
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_4 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_3 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 16388
=================================================================
Total params: 37,331,716
Trainable params: 37,328,964
Non-trainable params: 2,752
```

FIGURE 5.24: AlexNet Summary

Fig. 5.24 is the summary of AlexNet model. This is a handwritten AlexNet model using TensorFlow which has a total number of parameters levelling at 37 million. This is the reason AlexNet requires GPU to be trained in a reasonable amount of time.

5.7.3 AlexNet Execution and Results

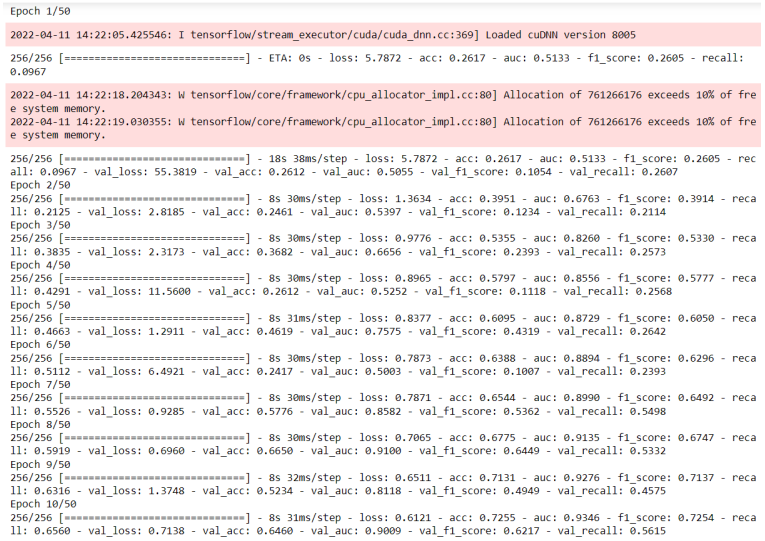

FIGURE 5.25: First 10 epochs of AlexNet

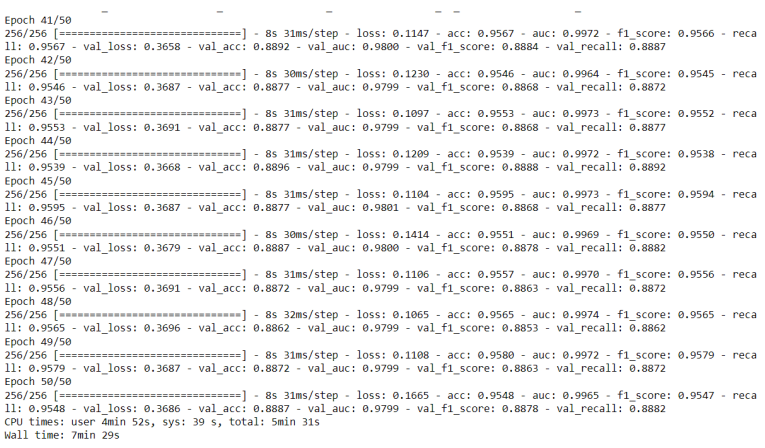

FIGURE 5.26: Last 10 epochs of AlexNet

Fig. 5.25 and Fig. 5.26 represent the first and last 10 epochs of training and validation of AlexNet respectively.

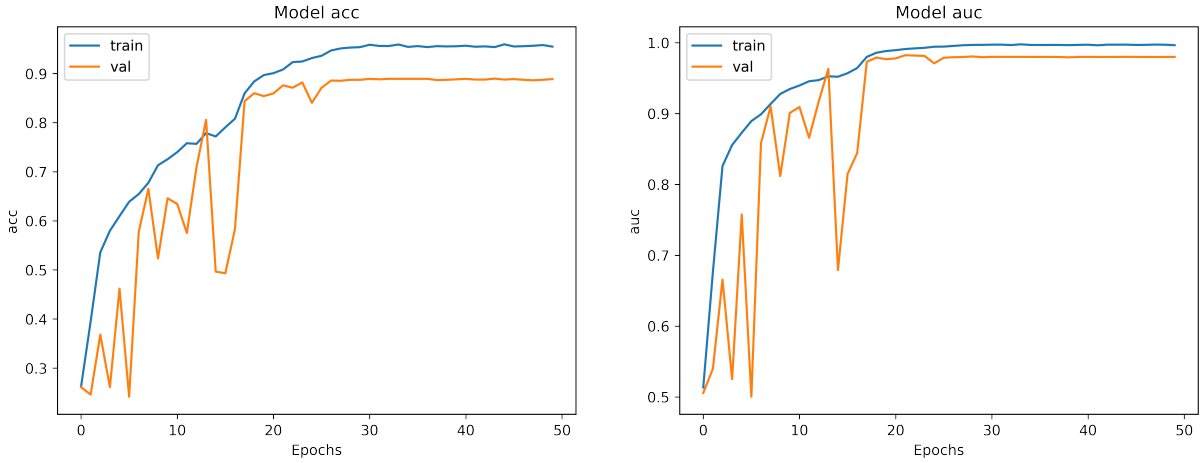

FIGURE 5.27: AlexNet accuracy and auc plot

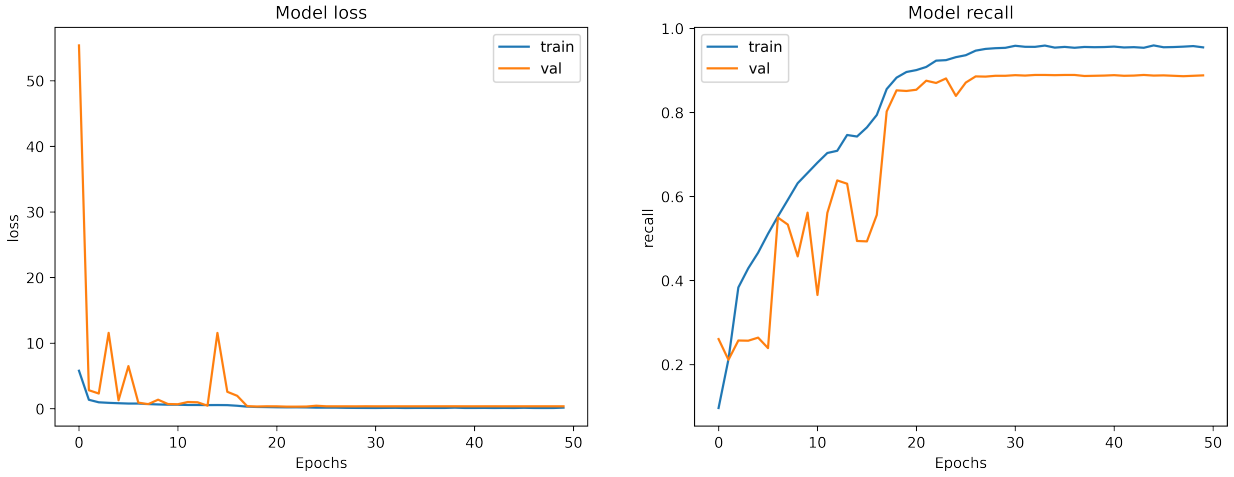

FIGURE 5.28: AlexNet loss and recall plot

From Fig. 5.27 and Fig. 5.28, It can be seen that the training accuracy rises steeply at first with the validation accuracy fluctuating a lot. However, both the metrics stabilize at the end of 50 epochs. The same trend is followed by AUCROC score and Recall metric. The loss is very high initially for the validation set but then quickly settles down to a low stable and desirable value.

5.8 WEIGHTED ENSEMBLE NETWORK

An ensemble of all the outputs of the various models is used here. We chose to preserve the probabilities a model produces for all 4 classes instead of consolidating it into a single answer. This is in stark contrast to the usual way ensemble is done. This ensures that each model's opinion on all 4 classes are taken into account to produce the final probabilities, rather than focussing on one definitive answer. Thus, we created a single layer neural network where the inputs are the probabilities of the various models and the output is a layer with 4 output nodes signifying the probabilities for each class. Since the ensemble learns which model is performing well and gives priority to it, the resulting accuracy and recall scores are noticeably higher than what is possible with individual models.

5.8.1 Weighed Ensemble Network Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, None, 4)           52
=================================================================
Total params: 52
Trainable params: 52
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 5.29: Ensemble Summary

Fig. 5.29 illustrates the summary of the weighted ensemble network encapsulating all the individual models which is also built upon the TensorFlow framework.

5.8.2 Weighted Ensemble Network Result

```
Epoch 25/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0479 - acc: 0.9838 - auc: 0.9994 - recall: 0.9827
Epoch 26/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0475 - acc: 0.9843 - auc: 0.9994 - recall: 0.9834
Epoch 27/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0472 - acc: 0.9840 - auc: 0.9994 - recall: 0.9835
Epoch 28/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0470 - acc: 0.9844 - auc: 0.9994 - recall: 0.9833
Epoch 29/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0468 - acc: 0.9844 - auc: 0.9994 - recall: 0.9835
Epoch 30/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0465 - acc: 0.9844 - auc: 0.9994 - recall: 0.9836
Epoch 31/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0464 - acc: 0.9845 - auc: 0.9994 - recall: 0.9840
Epoch 32/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0462 - acc: 0.9843 - auc: 0.9994 - recall: 0.9838
Epoch 33/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0460 - acc: 0.9845 - auc: 0.9994 - recall: 0.9841
Epoch 34/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0459 - acc: 0.9843 - auc: 0.9994 - recall: 0.9840
Epoch 35/50
256/256 [==============================] - 1s 4ms/step - loss: 0.0458 - acc: 0.9845 - auc: 0.9994 - recall: 0.9840
Epoch 36/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0457 - acc: 0.9844 - auc: 0.9994 - recall: 0.9839
Epoch 37/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0456 - acc: 0.9845 - auc: 0.9994 - recall: 0.9841
Epoch 38/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0456 - acc: 0.9845 - auc: 0.9994 - recall: 0.9843
Epoch 39/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0455 - acc: 0.9844 - auc: 0.9994 - recall: 0.9839
Epoch 40/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0454 - acc: 0.9849 - auc: 0.9995 - recall: 0.9845
Epoch 41/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0454 - acc: 0.9846 - auc: 0.9995 - recall: 0.9844
Epoch 42/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0453 - acc: 0.9844 - auc: 0.9995 - recall: 0.9840
Epoch 43/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0453 - acc: 0.9846 - auc: 0.9995 - recall: 0.9844
Epoch 44/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0453 - acc: 0.9845 - auc: 0.9995 - recall: 0.9840
Epoch 45/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0452 - acc: 0.9845 - auc: 0.9995 - recall: 0.9841
Epoch 46/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0452 - acc: 0.9845 - auc: 0.9995 - recall: 0.9844
Epoch 47/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0451 - acc: 0.9844 - auc: 0.9995 - recall: 0.9840
Epoch 48/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0451 - acc: 0.9843 - auc: 0.9995 - recall: 0.9840
Epoch 49/50
256/256 [==============================] - 1s 3ms/step - loss: 0.0451 - acc: 0.9844 - auc: 0.9995 - recall: 0.9841
Epoch 50/50
256/256 [==============================] - 1s 4ms/step - loss: 0.0451 - acc: 0.9844 - auc: 0.9995 - recall: 0.9844
```

FIGURE 5.30: Ensemble Results

Fig. 5.30 represents the last 25 epochs of executing the Weighted Ensemble Network.

5.9 RESULTS

The results of the individual models and the weighted ensemble network are given in the Table. 5.1 below.

Model	Loss	Accuracy	Area Under Receiver Operating Characteristics	Sensitivity
AlexNet	0.3685	0.8886	0.9799	0.8881
InceptionV3	0.2638	0.9003	0.9879	0.8979
DenseNet121	0.1962	0.9199	0.9926	0.9179
Weighted Ensemble Network	0.0451	0.9844	0.9995	0.9844

TABLE 5.1: Metrics table for individual models and the weighted ensemble network

Fig. 5.31 illustrates the confusion matrix of weighted ensemble network prediction using test data images. As can be viewed from the figure, the class *VeryMildDemented* has been predicted as such precisely with no mistakes. This means that there are no False Negatives for this class. The ensemble network faces little bit of confusion when predicting between classes *MildDemented* and *ModerateDemented* as seen in the figure.

5.10 DEPLOYMENT ON RASPBERRY PI

The ensemble network was offloaded to a Raspberry Pi 4 Model B, Fig. 5.32, to record the end-to-end inference time. This deployment was done using the *headless* approach. The Raspberry Pi was connected to the same wireless network

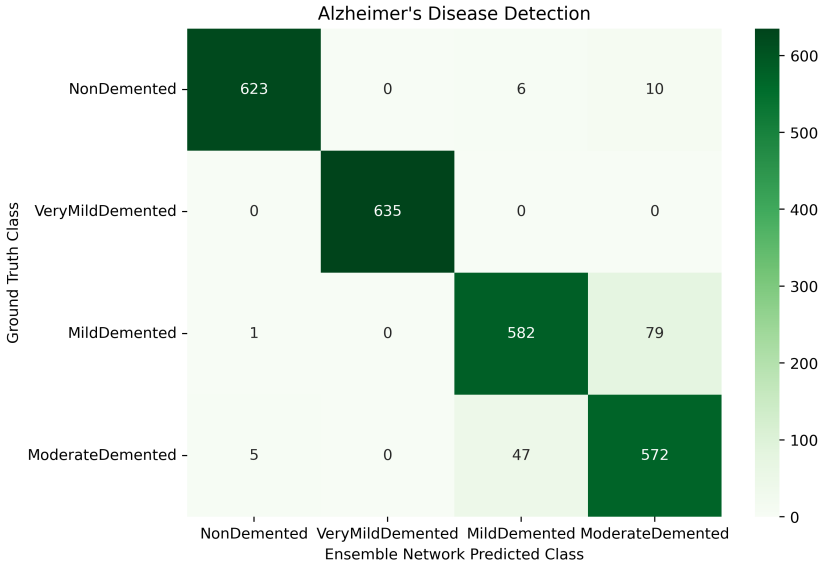

FIGURE 5.31: Confusion Matrix of test dataset

as the laptop, and the two devices were connected using **ssh**. Files were transferred to the edge-device using the service **scp** which stands for *Secure Copy*.

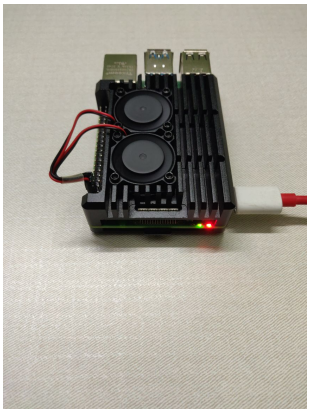

FIGURE 5.32: Deployed Raspberry Pi 4 Model B

5.11 SUMMARY

This chapter goes in detail about the implementation of the problem statement. The next chapter discusses and analyzes the obtained results.

CHAPTER 6

RESULT ANALYSIS AND DISCUSSION

In section, a profound analysis based on the results is discussed providing plausible reasons for the working of the weighted ensemble network.

6.1 COMPARISON BASED ON CLASS DISTRIBUTION

Table. 6.1 shows the distribution of samples under each class before and after balancing using SMOTE. It can be observed that, for the multi-class dataset, the number of samples in the minority classes has increased to that of the majority class.

Class	**Number of Samples before SMOTE**	**Number of Samples After SMOTE**
NonDemented	3200	3200
VeryMildDemented	2240	3200
MildDemented	896	3200
ModerateDemented	64	3200
Total	**6400**	**12800**

TABLE 6.1: Class Distribution before and after balancing

6.2 COMPARISON OF RESULTS WITH EXISTING WORK

Work	Accuracy	Sensitivity
Ocasio et. al [13]	0.79	-
Islam et. al [9]	0.9318	0.92
Weighted Ensemble Network	0.9844	0.9844
Performance increase	19% higher than [13] 5% higher than [9]	6.5% higher than [9]

TABLE 6.2: Comparison with existing research scores

The results of the individual models and the ensemble network is shown in Table. 6.2. There is a gradual increase in performance of the models from AlexNet through InceptionV3 to DenseNet-121. DenseNet with its densely coupled parameters showcases better scores overall. Combining the performance of the 3 models, the ensemble network has efficiently learned to select the best model suitable for an input and delivers significantly better scores than the other models. The ensemble network gives 98.4% accuracy when compared to Ocasio et. al's 79% accuracy [13]. Islam et. al's model renders 93.18% accuracy, 93% precision, 92% recall, and 92% F1-score, all of which are significantly lower than our ensemble network's performance [9].

6.3 MODEL METRICS

Table. 6.3 shows the model size and number of parameters of the three individual models and the ensemble network.

Models	Model Size	Total Number of Parameters
InceptionV3	93.1 MB	23, 036, 644
DenseNet121	33 MB	7, 742, 980
AlexNet	285 MB	37, 331, 716
Weighted Ensemble Network	18 KB	52

TABLE 6.3: Metrics of individual models

From the table above, it can be inferred that the AlexNet, due to its large number of parameters is of size 285MB, and InceptionV3 is considerably lesser in size coming at 93.1MB, while the DenseNet-121 is a third of that size at 33MB and only 7 million parameters. The total size of all the models and the network comes around 400MB which can comfortably fit an edge-device such as Raspberry Pi.

The ensemble network takes 12 input values (4 from each model) and consolidates it into 4 output values. So, there are $12 \times 4 = 48$ weights and 4 biases for each of the output nodes, thus totalling to 52 parameters.

6.4 INFERENCE TIME

```
(env) pi@raspberrypi:~/fyp $ python script.py
TensorFlow Version: 2.9.0
Loading data...
Loaded data!
Time to load all models = 38.68980073928833 seconds!
Beginning prediction...
1/1 [==============================] - 1s 1s/step
Time to predict using AlexNet = 2.1695382595062256 seconds!
1/1 [==============================] - 9s 9s/step
Time to predict using DenseNet121 = 11.318405866622925 seconds!
1/1 [==============================] - 6s 6s/step
Time to predict using InceptionV3 = 6.038468360900879 seconds!
1/1 [==============================] - 0s 96ms/step
Time to predict total = 19.703801155090332 seconds!
Time for single event prediction = 1.9703801155090332 seconds!
(env) pi@raspberrypi:~/fyp $
```

FIGURE 6.1: Inference Time Output

Fig. 6.1 depicts the output of loading all models, and using the ensemble to predict the output of a given input image. A script written to observe the inference time was executed 100 times on a sample set and the mean value was observed to be **1.97** seconds. This is a significantly short duration, as it is only a fraction of the time taken to produce an MRI brain scan. Therefore, this inference time is termed as **real-time**.

6.5 SUMMARY

Thus the number of samples in each class in the Alzheimer's Disease dataset before and after applying SMOTE is compared and tabulated. The properties of the individual models is noted. The next chapter concludes by giving an overview of the project, reasoning out the limitation, and potential future enhancements worthy of experimenting with.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

8In order to develop an automated pipeline for Alzheimer's Disease detection, a dataset containing MRI brain images was employed. GAN was utilized to generate artificial images to balance the dataset. However, the images created were not of sufficient detail to train the model. Hence, the class imbalance problem in the dataset was handled by applying SMOTE. Three individual models namely InceptionV3, DenseNet121 and AlexNet were trained separately on the training set. An ensemble network based on the weighted voting of the individual models was trained on the dataset. This network was transferred to an edge-device (RaspberryPi). After experimenting with the weighted ensemble network, the results recorded were 0.0451 loss, 98.44% accuracy, 99.95% AUC ROC score, and 98.44% sensitivity, as reported in Table. 5.1. This accuracy is 19% more than that of Ocasio et. al's Convolutional Neural Networks [10], and 5% more than that of Islam et. al's ensemble network of Dense Network [13]. Furthermore, loading the models onto Raspberry Pi takes 38 seconds, while a single prediction takes 1.97 seconds on an average, making it real-time. The limitation of this study is that GAN did not perform as intended. Different types of GAN's can be implemented and experimented with.

The possibility of using GAN for creating artificial images to balance datasets which are prone to imbalance like medical data can be explored further. This pipeline can be integrated with an MRI machine so as to render results in real-time. More models can be used as part of the ensemble network to increase the accuracy and sensitivity.

REFERENCES

1. Alroobaea, R., Mechti, S., Haoues, M., Rubaiee, S., Ahmed, A., Andejany, M., ... & Sengan, S. (2021). Alzheimer's Disease Early Detection Using Machine Learning Techniques.

2. Cha, D., Pae, C., Seong, S.-B., Choi, J. Y., & Park, H.-J. (2019). Automated diagnosis of ear disease using ensemble deep learning with a big otoendoscopy image database [Doi: 10.1016/j.ebiom.2019.06.050]. EBioMedicine, 45, 606–614. https://doi.org/10.1016/j.ebiom.2019.06.050

3. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357. https://doi.org/10.1613/jair.953

4. Dubey, Sarvesh. "Alzheimer's Dataset (4 Class of Images)." Kaggle, 26 Dec. 2019, https://www.kaggle.com/tourist55/alzheimers-dataset-4-class-of-images/data. (Accessed: 25 Oct 2021)

5. Gonzalez-Huitron, V., León-Borges, J. A., Rodriguez-Mata, A. E., Amabilis-Sosa, L. E., Ramírez-Pereda, B., & Rodriguez, H. (2021). Disease detection in tomato leaves via CNN with lightweight architectures implemented in Raspberry Pi 4. Computers and Electronics in Agriculture, 181, 105951. https://doi.org/https://doi.org/10.1016/j.compag.2020.105951

6. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. arXiv. https://doi.org/10.48550/ARXIV.1406.2661

7. Hosny. (2021). COVID-19 diagnosis from CT scans and chest X-ray images using low-cost Raspberry Pi. PLOS ONE, 16(5), 1–18. https://doi.org/10.1371/journal.pone.0250688

8. Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2016). Densely Connected Convolutional Networks. arXiv. https://doi.org/10.48550/ARXIV.1608.06993

9. Islam, J., & Zhang, Y. (2017). An Ensemble of Deep Convolutional Neural Networks for Alzheimer's Disease Detection and Classification. arXiv. https://doi.org/10.48550/ARXIV.1712.01675

10. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. Burges, L. Bottou, & K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems (Vol. 25). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

11. M. Sankar, D. N. Mudgal, T. varsharani jagdish, N. w. Geetanjali Laxman and M. Mahesh Jalinder, "Green Leaf Disease detection Using Raspberry pi," 2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT), 2019, pp. 1-6, doi: 10.1109/ICIICT1.2019.8741508.

12. Massalimova, A., & Varol, H. A. (2021). Input Agnostic Deep Learning for Alzheimer's Disease Classification Using Multimodal MRI Images. arXiv preprint arXiv:2107.08673.

13. Ocasio, E., & Duong, T. Q. (2021). Deep learning prediction of mild cognitive impairment conversion to Alzheimer's disease at 3 years after diagnosis using longitudinal and whole-brain 3D MRI. PeerJ Computer Science, 7, e560.

14. Pan, D., Zeng, A., Jia, L., Huang, Y., Frizzell, T., & Song, X. (2020). Early Detection of Alzheimer's Disease Using Magnetic Resonance Imaging: A Novel Approach Combining Convolutional

Neural Networks and Ensemble Learning. Frontiers in Neuroscience, 14. https://doi.org/10.3389/fnins.2020.00259

15. Perez, Luis, and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621 (2017).

16. Qiu, C., Kivipelto, M., & von Strauss, E. (2009). Epidemiology of Alzheimer's disease: occurrence, determinants, and strategies toward intervention. Dialogues in clinical neuroscience, 11(2), 111–128. https://doi.org/10.31887/DCNS.2009.11.2/cqiu

17. Ruiz, Juan & Mahmud, Mufti & Modasshir, Md & Kaiser, M. Shamim & Initiative, for. (2020). 3D DenseNet Ensemble in 4-Way Classification of Alzheimer's Disease. 10.1007/978-3-030-59277-6_8

18. Schachter, A. S., & Davis, K. L. (2000). Alzheimer's disease. Dialogues in clinical neuroscience, 2(2), 91–100. https://doi.org/10.31887/DCNS.2000.2.2/asschachter

19. Sharma, J., & Kaur, S. (2017, August). Gerontechnology—The study of Alzheimer's disease using cloud computing. In 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS) (pp. 3726-3733). IEEE.

20. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. arXiv. https://doi.org/10.48550/ARXIV.1512.00567

21. Wang, Y., Liu, X., & Yu, C. (2021). Assisted Diagnosis of Alzheimer's Disease Based on Deep Learning and Multimodal Feature Fusion. Complexity, 2021.

22. Wu, J. M.-T., Tsai, M.-H., Huang, Y. Z., Islam, S. H., Hassan, M. M., Alelaiwi, A., & Fortino, G. (2019). Applying an ensemble

convolutional neural network with Savitzky–Golay filter to construct a phonocardiogram prediction model. Applied Soft Computing, 78, 29–40. https://doi.org/https://doi.org/10.1016/j.asoc.2019.01.019

23. Yamanakkanavar, N., Choi, J. Y., & Lee, B. (2020). MRI segmentation and classification of human brain using deep learning for diagnosis of Alzheimer's disease: a survey. Sensors, 20(11), 3243.

24. Zhou, X., Qiu, S., Joshi, P. S., Xue, C., Killiany, R. J., Mian, A. Z., ... & Kolachalama, V. B. (2021). Enhancing magnetic resonance imaging-driven Alzheimer's disease classification performance using generative adversarial learning. Alzheimer's research & therapy, 13(1), 1-11.

25. "Dementia." World Health Organization, World Health Organization, https://www.who.int/news-room/fact-sheets/detail/dementia. (Accessed: 24 May 2022)