PROJECT DOCUMENTATION

ALZHEIMER DISEASE PREDICTION

Index

1. Introduction

1.1 Project overview

This project aims to provide users with an accurate setup to predict Alzheimer's disease and its stages. This model was developed using CNN algorithms with the help of the dataset collected using python programming language. The primary purpose of this project is to increase the accuracy and efficiency of prediction of Alzheimer's disease using deep learning algorithms. For this purpose, we have used a self-evolving model which on usage updates and improvises on its own. Thus giving maximum efficiency and accuracy.

1.2 Purpose

The sole objective of this project is to create a model that processes MRI images using CNN to find out Alzhiemer's disease by analysing the MRI pattern in patients and predict the stage of illness as well as the potential chances of being affected by the disease in near future. Thereby this project aims at helping patients as well as caregivers get closer attention to the illness and help them take the necessary step at the earliest and help them lead a better and happier life.

2. Literature Survey

2.1 Existing Problem

Alzheimer's disease, an advancing neurological disorder, presents a significant hurdle in healthcare due to the limitations of its detection methods. Despite thorough research, several key issues remain, impeding timely and accurate diagnosis.

The absence of a definitive diagnostic test for Alzheimer's is a critical concern. Existing diagnostic methods depend on clinical evaluations and cognitive assessments, which are subjective and prone to discrepancies between evaluators. This subjectivity leads to a delay in diagnosis, thereby reducing the effectiveness of potential treatments.

Additionally, while neuroimaging techniques like MRI and PET scans provide valuable information, they are expensive, resource-intensive, and not accessible to a large population of patients. Furthermore, the interpretation of these images can be complex, requiring specialized expertise.

The variability of Alzheimer's disease further complicates the detection process. The disease manifests differently in different individuals, making it challenging to establish a universal diagnostic criterion. Subtypes of Alzheimer's, such as early-onset or atypical forms, require specific detection methods, adding to the complexity of the diagnostic process.

To tackle these challenges, collective efforts are required to develop more objective, cost-effective, and widely accessible diagnostic tools. Emerging technologies like machine learning and biomarker research show promise in improving the detection of Alzheimer's

disease. However, overcoming these challenges is crucial to enable early intervention, optimize patient care, and advance research towards finding a cure.

Various deep learning models have been applied to detect Alzheimer's using brain MRI images. Different models, such as convolutional neural networks (CNNs) and deep learning-based segmentation methods, show varying levels of accuracy in differentiating Alzheimer's brains from healthy brains. These models use advanced techniques like transfer learning and have shown promising results, with accuracy rates ranging from 73.75% to 100%. The use of deep learning in analyzing medical data, including functional MRI, is seen as a potential game-changer in the early diagnosis and classification of Alzheimer's disease.

2.2 References

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8083897/
https://philarchive.org/archive/SAMCOA-4
https://www.mdpi.com/2075-4418/11/3/440
https://www.sciencedirect.com/science/article/abs/pii/S193152441830001X
https://link.springer.com/article/10.1007/s10462-021-10016-0

2.3 Problem statement

Alzheimer's disease (AD) is a progressive and irreversible neurological disorder that affects the brain, leading to memory loss, cognitive deterioration, and changes in behavior and personality. It is the leading cause of dementia in the elderly and is marked by the build-up of abnormal protein deposits, including amyloid plaques and tau tangles. The exact cause of Alzheimer's is not fully understood, but it is believed to be influenced by a mix of genetic, environmental, and lifestyle factors. Age is a major risk factor, with the probability of developing the disease increasing significantly after the age of 65.

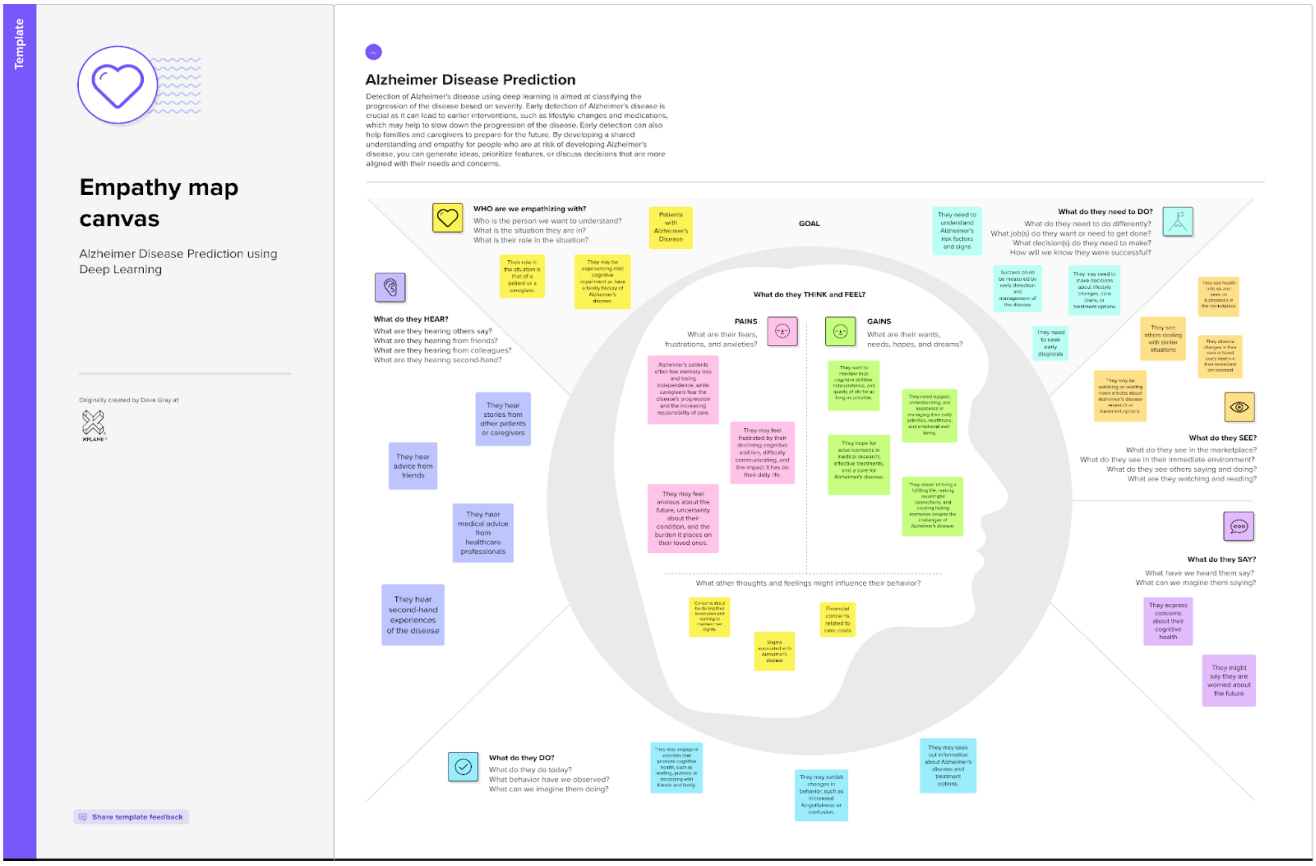
The early symptoms of Alzheimer's may include mild memory loss, difficulties in problem-solving, and changes in mood or behavior. As the disease progresses, these symptoms become more severe, resulting in significant memory loss, impaired communication, and the inability to carry out daily activities.

The use of deep learning models like Xception to analyze medical imaging data has the potential to detect early signs of Alzheimer's before the symptoms become severe. This method allows healthcare providers to intervene and provide support to patients and their families, ultimately improving outcomes for all involved. After predicting the stage of the disease, the associated web application also suggests appropriate prognoses based on the current progression of the disease and the individual's health situation.

3. Ideation and Proposed Solution

3.1 Empathy Mapping Canvas

The purpose of this phase was to empathise with the patients and caregivers as the name suggests. This step included empathising with their feelings, fears, anxieties, challenges and hopes.
This stage broadly discussed and debated the actual needs and challenges faced by the targeted community, which includes what they feel, hear, speak, think,do and their gains and losses. Empathy mapping is the foundation on which the project objectives will be developed in the later stages.



3.2 Ideation and Brainstorming

The objective of this phase is to ideate the required solution for the uptaken challenge. The problem statement is to develop an end-to-end framework for early stage detection of Alzheimer's disease and for various medical image classification for various AD stages. As empathising was followed by the stages of ideation and brainstorming we could easily connect with the beneficiaries and develop as per their requirements.
In the brainstorming stage, each team member came up with their ideas on how the project must be executed, its prime objectives and other necessary information. These ideas were broadly discussed and the most vital, viable and feasible ones were finalised. The finalised objectives were sorted and classified based on their feasibility and importance.

1 Define your problem statement

The Alzheimer's disease prediction project aims to design an end-to-end framework for early detection of Alzheimer's disease and medical image classification for various AD stages. Early detection of Alzheimer's disease is crucial as it can lead to earlier interventions, such as lifestyle changes and medications, which may help to slow down the progression of the disease. By developing a shared understanding and empathy for people who are at risk of developing Alzheimer's disease, you can generate ideas, prioritize features, or discuss decisions that are more aligned with their needs and concerns.

⏱ 5 minutes

PROBLEM

How might we [your problem statement]?

Key rules of brainstorming

To run an smooth and productive session

- Stay in topic.
- Defer judgment.
- Go for volume.
- Encourage wild ideas.
- Listen to others.
- If possible, be visual.

2 Brainstorm

Write down any ideas that come to mind that address your problem statement.

⏱ 10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Jithu

- A web application using Unsupervised Learning Techniques
- Develop a convolutional neural network (CNN) model for predicting stages.
- Collection of Dataset from ADNI, OASIS or NIAGADS

Arun

- Integration of brain imaging data for early Alzheimer's detection.
- Create a user-friendly mobile app that allows individuals to upload images for disease prediction and provides educational resources on Alzheimer's disease.
- NLP techniques analyzing speech patterns for early detection.

Saathwick

- Using deep learning techniques like ResNet and AlexNet
- Using deep learning to analyze speech and vocabulary pattern
- Using CNN to analyze MRI pattern of patients

Mahaashwanth

- Analyzing white matter changes and lesions in MRI scans can indicate cognitive decline in Alzheimer's disease.
- Longitudinal analysis of data to track disease progression and predict outcomes.
- Facilitate remote patient monitoring through wearable devices and remote image acquisition to streamline charge detection and tracking.
- Augment MRI data using techniques like rotation, noise, or resolution to improve the model's robustness and ability to handle diverse scan qualities.

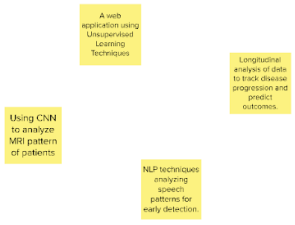
3 Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

⏱ 20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.



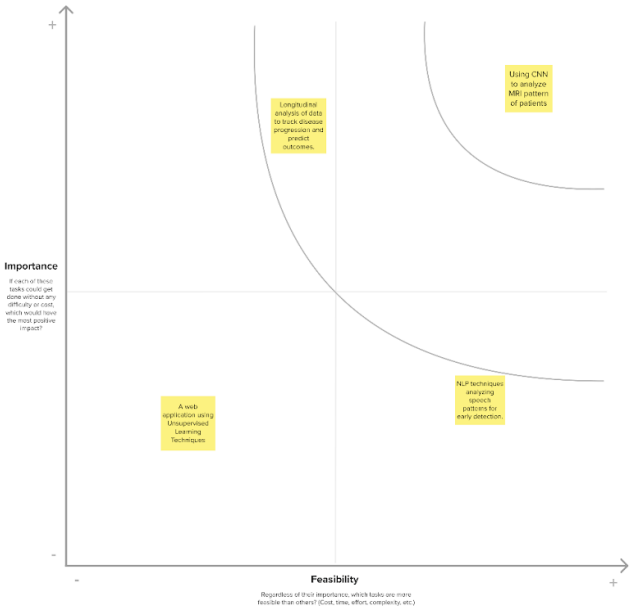
4 Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

⏱ 20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H** key on the keyboard.



4. Requirement Analysis

4.1 Functional Requirements

1. **Data Preprocessing:** The system should possess the capability to preprocess and cleanse extensive neuroimaging data, encompassing MRI and PET scans. This includes extracting pertinent features essential for Alzheimer's disease prediction.
2. **Model Architecture Implementation:** Implement the deep learning model for image classification to precisely forecast Alzheimer's disease based on neuroimaging data.
3. **Training and Evaluation:** The system must be proficient in training the model using the preprocessed data. Additionally, it should be able to assess the model's performance using relevant metrics such as accuracy, precision, recall, and F1 score.
4. **User Interface:** Develop an intuitive and user-friendly interface that facilitates researchers and medical professionals in seamlessly uploading, processing, and predicting Alzheimer's disease from new neuroimaging data.

4.2 Non-functional Requirements

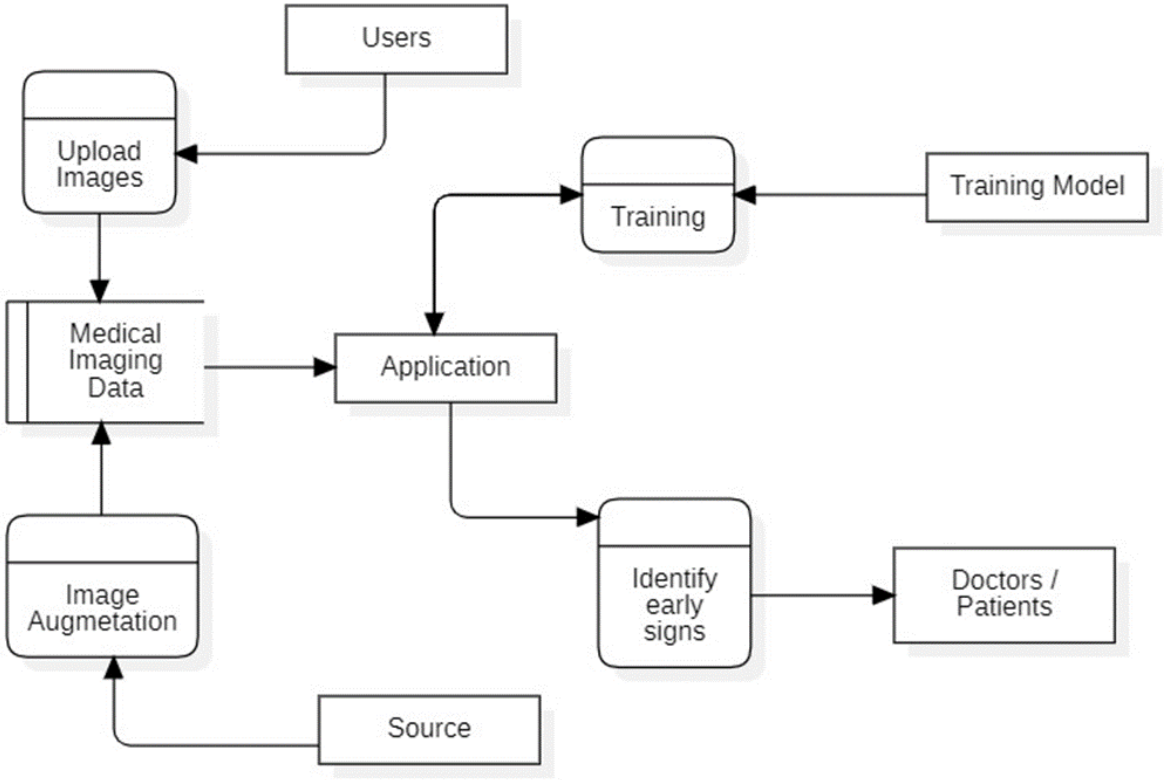
1. **Performance**: The model is expected to deliver a high level of accuracy (over 90%) and provide predictions on time, ensuring its suitability for use in clinical settings.
2. **Scalability**: The system needs to be designed to manage a substantial amount of neuroimaging data and should be flexible enough to accommodate future dataset expansions and updates.
3. **Security**: The implementation of robust security measures such as encryption, access controls, and user authentication is crucial to ensure the protection of sensitive patient data and compliance with privacy regulations.
4. **Robustness**: The system should be resilient enough to handle variations in the quality and size of input data, providing consistent and reliable predictions even when the neuroimaging data contains noise or artifacts.
5. **Usability**: The interface should be intuitive and user-friendly, requiring minimal training for users to upload data, interpret results, and comprehend the model’s predictions.

5. Project Design

5.1 Data Flow Diagram and User Stories

5.1.1 Data flow diagram

Data flow diagram (DFD) is a pictorial representation of the workflow. A well crafted DFD gives a clear insight on the system requirements data flow, data management, manipulations and storage.



5.1.2 User Stories

User stories depict the difficulties faced, the grievances expressed and the requirements unfulfilled as per the feedback given by the users of the product. Here the users refer to patients and caregivers and other medical practitioners who interact with Alzheimer's patients and their well-being. The feedback collected at various stages of the product development phases from beneficiaries at different levels are listed below.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Healthcare professionals	Project setup & infrastructure	USN-1	Set up the necessary infrastructure for the Alzheimer's disease prediction project using deep learning.	Infrastructure is successfully set up for the project.	Medium	Sprint 1

Researchers	Development environment	USN-2	Set up the development environment with the required tools and frameworks to start the Alzheimer's disease prediction project using deep learning.	The development environment is properly configured and ready for use.	Medium	Sprint 1
Data scientists	Data collection	USN-3	Gather a diverse dataset of medical imaging data, genetic information, and cognitive assessments for training the deep learning model for Alzheimer's disease prediction.	Sufficient and representative data is collected for training the model.	HIgh	Sprint 2
Neurologists	Data preprocessing	USN-4	Preprocess the collected dataset by cleaning, normalizing, and transforming the data to make it suitable for training the deep learning model.	Data is cleaned, normalized, and prepared for training.	High	Sprint 2
Patients and Caregivers	Model development	USN-5	Explore and evaluate different deep learning architectures (e.g., CNNs, RNNs) to select the most suitable model for Alzheimer's disease prediction.	Deep learning model is developed and ready for training.	High	Sprint 3
Medical Institutions	Training	USN-6	Train the selected deep learning model using the preprocessed dataset and monitor its performance on a validation set.	Model is trained on the data and achieves satisfactory performance.	High	Sprint 3

	Model deployment & integration	USN-7	Deploy the trained deep learning model as an API or web service to make it accessible for Alzheimer's disease prediction. Integrate the model's API into a user-friendly web interface for users to input relevant data and receive Alzheimer's disease prediction results.	Model is successfully deployed and integrated into the system.	Medium	Sprint 4
	Testing & quality assurance	USN-8	Conduct thorough testing of the model and web interface to identify and report any issues or bugs. Fine-tune the model's hyperparameters and optimize its performance based on user feedback and testing results.	Model passes all tests and meets the required quality standards.	Medium	Sprint 4

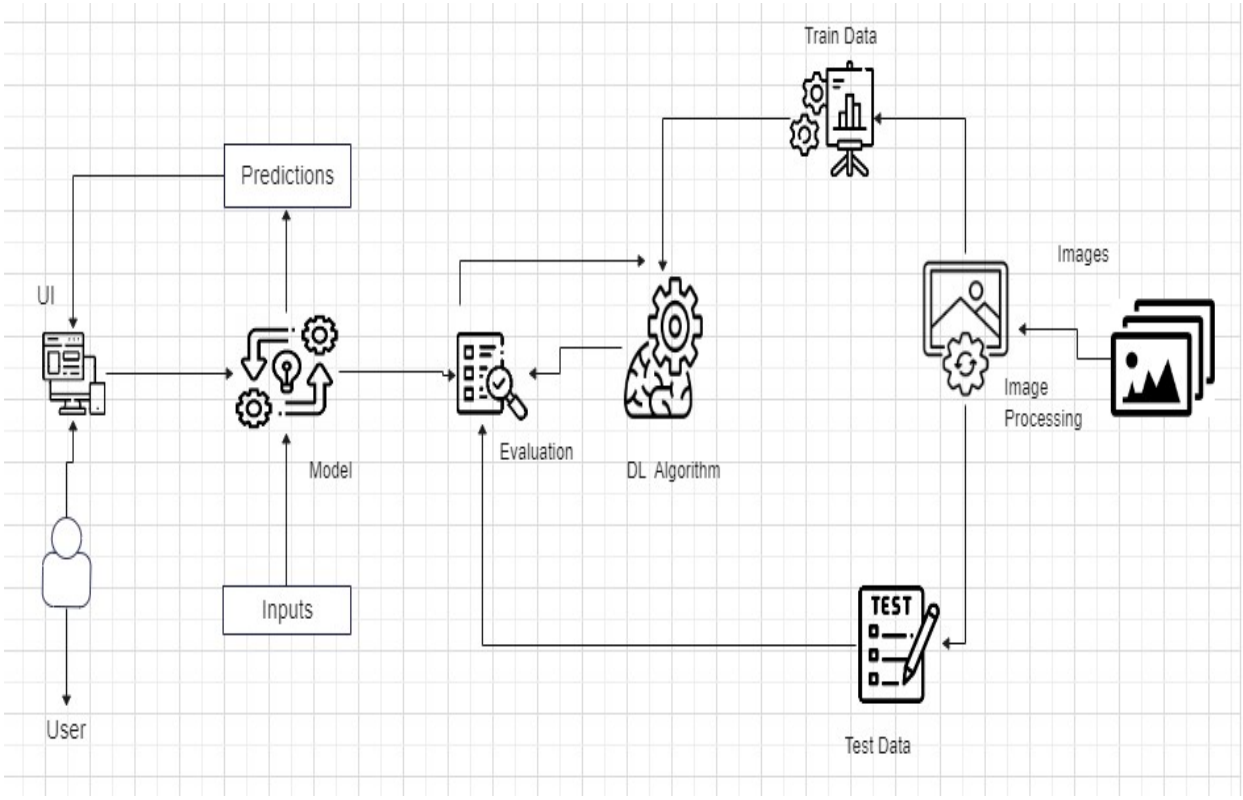
5.2 Solution Architecture

It optimizes and enhances the process of predicting Alzheimer's in patients in a very early stage with the help of CNNs real time image classification algorithm. It altogether enhances the accuracy and efficiency of predicting the disease at a very early stage itself helping in early diagnosis and treatment which helps to make the life a lot more easier and happier for both patients and care-givers.This infinite closed-loop system ensures real-time updates in the brain pattern of subjects using regular updates of MRI pattern recognition leading ever evolving accuracy and efficiency.

Once the model is built, it is saved and its predictions are used as input to a website created via Flask deployment, which then provides the resultant treatment strategy. The multiple layers of the CNN model contribute to effective model training. Our solution

effectively addresses the Alzheimer's disease prediction problem by leveraging a deep learning strategy that uses pre-trained CNN models.

- **Data Gathering**: We utilize an image dataset obtained from Kaggle, which includes images representing different stages of Alzheimer's disease progression.
- **Image Preprocessing**: The input images will be resized based on the requirements of the individual model. The images will then be flattened to input the pooling features into the neural network's input layer.
- **Model Building**: We employ a deep learning strategy which uses CNN model for training and prediction.
- **Alzheimer's Disease Prediction**: After testing the model for its accuracy and errors through hyperparameter tuning, select the best one for our use case and dataset.
- **Real-Time Analysis**: Finally, we will download the model and create a web application using Flask. This will provide a user interface for users to use our model and check for Alzheimer's by uploading their EMR image to the website.



6. Project Planning and Scheduling

6.1 Technology Stack

Technical Architecture

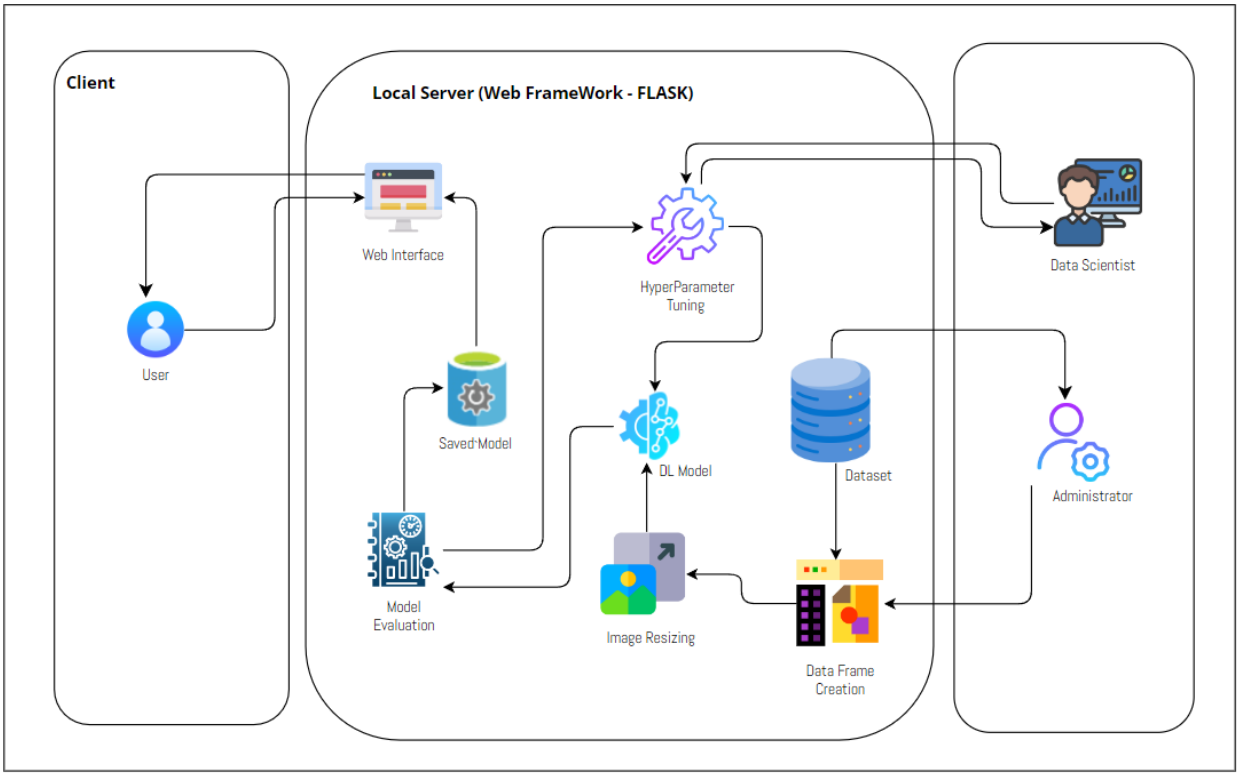


Table-1 : Components & Technologies

S.No.	Component	Description	Technology
1	User Interface	Users can upload input images to the website and use the predict option in the website to predict the current stage of Alzheimer.	HTML, CSS, JavaScript.
2	Application Logic-1	Downloading the dataset, dataframe creation, preprocessing.	Python (pandas, numpy)
3	Application Logic-2	Model building, Model evaluation and Hyperparameter Tuning.	Python (Tensorflow)
4	Database	Image Database. (jpeg)	File Manager
5	File Storage	All files saved locally.	File Manager

6	External API-1	Flask provides an API that simplifies the development of RESTful web services by offering features like Routes and Request Objects. It enables developers to access details about the incoming HTTP request, including the URL, request method, and request data.	Flask
7	Deep Learning Model	The transfer learning model uses deep learning techniques which reads image datasets and trains them based on CNN layers for accurate prediction of result.	Custom Sequential CNN

Table-2 : Application Characteristics

S.No.	Characteristics	Description	Technology
1	Open-Source Frameworks	Flask is a lightweight Python web framework known for its simplicity and ease of use a great choice for building small to medium-sized web applications.	Flask FrameWork
2	Scalable Architecture	3 tier Client-Server Architecture using a local host.	Visual Paradigm
3	Availability	The application will be available in the local environment.	Local Host
4	Performance	Used by a single user at a time.	Command Line

6.2 Sprint Planning and Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint 1	Project setup & Development Environment	USN-1	Set up the necessary infrastructure for the Alzheimer's disease prediction project using deep learning.	3	Medium	Arun
Sprint 1	Project setup & Development Environment	USN-2	Set up the development environment with the required tools and frameworks to start the Alzheimer's disease prediction project using deep learning.	2	Medium	Jithu
Sprint 2	Data Collection and Preprocessing	USN-3	Gather a diverse dataset of medical imaging data, genetic information, and cognitive assessments for training the deep learning model for Alzheimer's disease prediction.	5	High	Saathwick

Sprint 2	Data Collection and Preprocessing	USN-4	Preprocess the collected dataset by cleaning, normalizing, and transforming the data to make it suitable for training the deep learning model.	5	High	Mahaashwath
Sprint 3	Model Development and Training	USN-5	Explore and evaluate different deep learning architectures (e.g., CNNs, RNNs) to select the most suitable model for Alzheimer's disease prediction.	7	High	Arun
Sprint 3	Model Development and Training	USN-6	Train the selected deep learning model using the pre-processed dataset and monitor its performance on a validation set.	8	High	Jithu
Sprint 4	Model Deployment and Testing	USN-7	Deploy the trained deep learning model as an API or web service to make it accessible for Alzheimer's disease prediction.	3	High	Mahaashwath

Sprint 4	Model Deployment and Testing	USN-8	Integrate the model's API into a user-friendly web interface for users to input relevant data and receive Alzheimer's disease prediction results.	3	Medium	Jithu
Sprint 4	Model Deployment and Testing	USN-9	Conduct thorough testing of the model and web interface to identify and report any issues or bugs. Fine-tune the model's hyperparameters and optimize its performance based on user feedback and testing results.	2	Medium	Saathwick
Sprint 4	Model Deployment and Testing	USN-10	Fine-tune the model's hyperparameters and optimize its performance based on user feedback and testing results.	2	Medium	Arun

6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint 1	5	1 Day	07 Nov 2023	08 Nov 2023	5	08 Nov 2023

Sprint 2	10	1 Day	08 Nov 2023	09 Nov 2023	15	09 Nov 2023
Sprint 3	15	4 Days	09 Nov 2023	12 Nov 2023	25	12 Nov 2023
Sprint 4	10	3 Days	12 Nov 2023	14 Nov 2023	35	14 Nov 2023

7. Coding & Solutioning

Project Flow

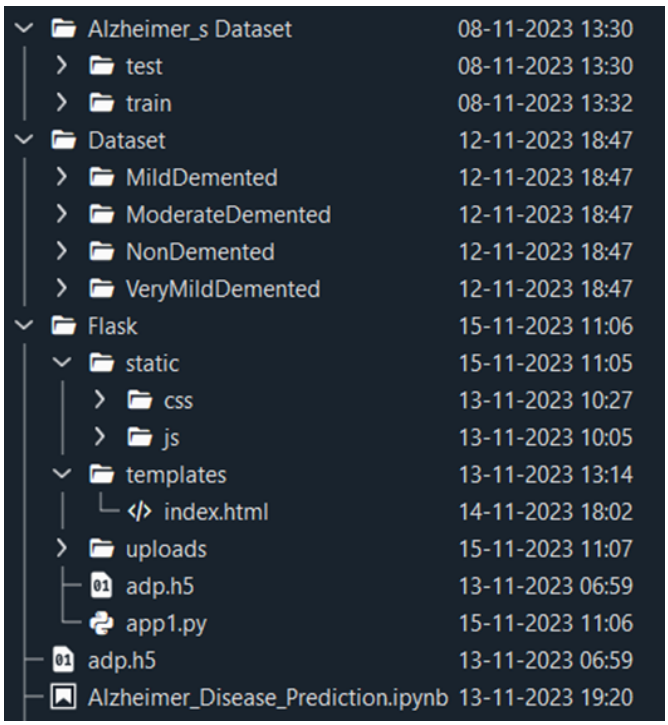
- The user interacts with the UI to choose an image.
- The chosen image is processed by a deep learning model.
- The model is integrated with a Flask application.
- The model analyzes the image and generates predictions.
- The predictions are displayed on the Flask UI for the user to see.
- This process enables users to input an image and receive accurate predictions quickly.

To accomplish this, we have to complete all the activities and tasks listed below

- ➢ Data Collection.
  - A Kaggle image dataset consisting of scanned images of the brain is used
  - Create a Train and Test path.
- ➢ Image Pre-processing.
  - Import the required libraries
  - Configure Image Data Generator class
  - Handling imbalance data
  - Splitting into train-test split
- ➢ Model Building
  - CNN model as a Feature Extractor
  - Creating Sequential layers
  - Configure the Learning Process
  - Train the model
  - Save the Model
  - Test the model
- ➢ Application Building
  - Create an HTML file
  - Build Python Flask Code for creating and accessing the prediction website to obtain accurate diagnosis of Alzheimer's disease upon request.
  - Run the application.

Project Structure

Create a Project folder which contains files as shown below



- The Alzheimer_s Dataset folder contains the test and train folders downloaded from Kaggle.
- The Dataset folder contains the rearranged images from the Alzheimer Dataset into 4 folders training our model.
- For building a Flask Application we needs HTML pages stored in the templates folder, CSS for styling the pages stored in the static folder and a python script app1.py for server side scripting.
- The folder consists of Alzheimer_Disease_Prediction.ipynb - model training file & adp.h5 – which is the saved model.

Milestone 1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

Activity 1: Download the dataset

Collect images of brain MRI then organize into subdirectories based on their respective names as shown in the project structure. Create folders of types of Alzheimer's.

In this project, we have collected images of 4 types of brain MRI images like Mild Demented, Moderate Demented, Non Demented & Very Mild Demented and they are saved in the respective sub directories with their respective names.

You can download the dataset used in this project using the below link

Dataset: Alzheimer's Dataset (4 class of Images) | Kaggle

Activity 2: Create a new Dataset

```
import os
import shutil

# Define the source and destination directories
source_dir = r"C:\Users\chris\Downloads\ADP Copy 1\Alzheimer_s Dataset"
destination_dir = r"C:\Users\chris\Downloads\ADP Copy 1\Dataset"

# Create the destination directory if it doesn't exist
if not os.path.exists(destination_dir):
    os.makedirs(destination_dir)

# Define the folder names
folders = ["VeryMildDemented", "ModerateDemented", "NonDemented", "MildDemented"]

# Loop through the folders
for folder in folders:
    # Create the corresponding folder in the destination directory
    destination_folder = os.path.join(destination_dir, folder)
    if not os.path.exists(destination_folder):
        os.makedirs(destination_folder)

    # Loop through the train and test folders
    for subfolder in ["train", "test"]:
        subfolder_path = os.path.join(source_dir, subfolder, folder)

        # Check if the subfolder exists
        if os.path.exists(subfolder_path):
            # Loop through the files in the subfolder
            for file_name in os.listdir(subfolder_path):
                file_path = os.path.join(subfolder_path, file_name)

                # Copy the file to the corresponding folder in the destination directory
                destination_file_path = os.path.join(destination_folder, file_name)
                shutil.copy(file_path, destination_file_path)
```

Milestone 2: Image Processing

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Import Necessary Libraries

Import important libraries like TensorFlow, keras, NumPy, pandas, os, imblearn, sklearn and so on. Also import the necessary modules.

```
import numpy as np
import pandas as pd
import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

from tensorflow.keras import Sequential,Input
from tensorflow.keras.layers import Conv2D, Dropout
from tensorflow.keras.layers import BatchNormalization, MaxPool2D
from tensorflow.keras.layers import Dense, Flatten
import tensorflow_addons as tfa

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

Activity 2: Configure ImageDataGenerator class

The ImageDataGenerator class is part of the tensorflow.keras.preprocessing.image module and is used for generating batches of augmented image data for training a neural network. This additional set of modified images provides additional batches of input for training which improves the capability of the model. The image size is set to (176,176) before being augmented. Here's a breakdown of the parameters used in this example:

- **rescale**=1./255: This normalizes the pixel values of the image to the range of [0, 1], which is a common practice in deep learning models.
- **zoom_range**=[0.99,1.01]: This applies random zooming transformations to the image, which can help the model learn to be more robust to different scales of objects in the image.
- **brightness_range**=[0.8,1.2]: We can use it to adjust the brightness_range of any image for Data Augmentation.
- **horizontal_flip**=True: This applies random horizontal flipping to the image, which can help the model learn to be more invariant to the orientation of objects in the image.
- **fill_mode**='constant': 'fill_mode' is a parameter used to specify the strategy for filling in newly created pixels that might appear after a transformation. When it is 'constant', any newly created pixels outside the boundaries of the original image are filled by a constant value specified by the 'cval' parameter which is 0 by default.
- **data_format** = 'channels_last': 'data_format' is a parameter that determines the ordering of the dimensions in the input data. It specifies how the data is structured in multi-dimensional arrays.When it is set as 'channels_last', the order for 2D case is (samples, height, width, channels) and for 3D case is (samples, depth, height, width, channels).
- Also set the batch_size as 6500 and shuffle as False and assign the correct directories for train and test.
- These data augmentation techniques can help increase the diversity and size of the training data, which can improve the performance of the model and reduce overfitting. You can use the train_data_gen object to generate batches of augmented training data on the fly, as needed by the fit() method of a Keras model.

```
#2. Configure image data generator
IMG_SIZE = 176
IMAGE_SIZE = [176,176]
DIM = (IMG_SIZE, IMG_SIZE)

ZOOM = [.99, 1.01]
BRIGHTNESS = [0.8, 1.2]
FILL_MODE = "constant"
DATA_FORMAT = "channels_last"
train_datagen = ImageDataGenerator(rescale = 1./255, brightness_range=BRIGHTNESS, zoom_range=ZOOM, data_format=DATA_FORMAT, fill_
```

```
#3. Apply image data generator functionality to train and test images
x_train = train_datagen.flow_from_directory(r"C:\Users\chris\Downloads\ADP Project\Dataset",target_size=DIM, batch_size=6500, shu
```

```
Found 6400 images belonging to 4 classes.
```

```
train_data, train_labels = x_train.next()
print(train_data.shape, train_labels.shape)
```

```
(6400, 176, 176, 3) (6400, 4)
```

```
print(x_train.class_indices)
```

```
{'MildDemented': 0, 'ModerateDemented': 1, 'NonDemented': 2, 'VeryMildDemented': 3}
```

Activity 3: Handling Imbalance Data (SMOTE)

- The SMOTE() is used to apply the SMOTE oversampling technique to balance the dataset. The random_state value provided is 42.
- The reshape(-1, IMG_SIZE*IMG_SIZE*3) is used to reshape the 2D matrix shape of the input image of (176,176,3) dimension to a 1D array of length 176*176*3.
- The sm.fit_resample technique is used to apply the SMOTE technique on the reshape train_dataset. After the sampling is applied the images are again reshaped back to a 4D array of shape (batch_size, IMG_size, IMG_size,3).
- After oversampling the data which had a shape of (6400, 176, 176, 3) has now doubled to (12,800, 176, 176, 3). The data has been doubled due to oversampling. As we can observe, from 6400 to 12800.

```
#Performing over-sampling of the data, since the classes are imbalanced
sm = SMOTE(random_state=42)
train_data, train_labels = sm.fit_resample(train_data.reshape(-1, IMG_SIZE * IMG_SIZE * 3), train_labels)
train_data = train_data.reshape(-1, IMG_SIZE, IMG_SIZE, 3)
print(train_data.shape, train_labels.shape)
```

```
(12800, 176, 176, 3) (12800, 4)
```

Activity 4: Splitting into train test split

Now we are splitting the labels into train-test split in 80:20 ratio & assigning them to train_data, train_labels & test_data, test_lables.

Validation sets of train_data, val_data, train_labels, val_labels.

```
#Splitting the data into train, test, and validation sets
train_data, test_data, train_labels, test_labels = train_test_split(train_data, train_labels, test_size = 0.2, random_state=42)
train_data, val_data, train_labels, val_labels = train_test_split(train_data, train_labels, test_size = 0.2, random_state=42)
```

Milestone 3: Model Building

Activity 1: Constructing a CNN Architecture

Define Convolution NN and Dense NN block for Sequential CNN model.

```
def conv_block(filters, act='relu'):
    """Defining a Convolutional NN block for a Sequential CNN model. """

    block = Sequential()
    block.add(Conv2D(filters, 3, activation=act, padding='same'))
    block.add(Conv2D(filters, 3, activation=act, padding='same'))
    block.add(BatchNormalization())
    block.add(MaxPool2D())

    return block
```

```
def dense_block(units, dropout_rate, act='relu'):
    """Defining a Dense NN block for a Sequential CNN model. """

    block = Sequential()
    block.add(Dense(units, activation=act))
    block.add(BatchNormalization())
    block.add(Dropout(dropout_rate))

    return block
```

Define a construct_model function block for constructing a Sequential CNN architecture for performing the classification task.

```
def construct_model(act='relu'):
    """Constructing a Sequential CNN architecture for performing the classification task. """

    model = Sequential([
        Input(shape=(*IMAGE_SIZE, 3)),
        Conv2D(16, 3, activation=act, padding='same'),
        Conv2D(16, 3, activation=act, padding='same'),
        MaxPool2D(),
        conv_block(32),
        conv_block(64),
        conv_block(128),
        Dropout(0.2),
        conv_block(256),
        Dropout(0.2),
        Flatten(),
        dense_block(512, 0.7),
        dense_block(128, 0.5),
        dense_block(64, 0.3),
        Dense(4, activation='softmax')
    ], name = "cnn_model")

    return model
```

Activity 2: Configure the Learning Process

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.

Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. In this case, we are using the 'adam' optimizer.

Metrics are used to evaluate the performance of your model. They provide insights into how well the model is performing but are not used during the training process. In our code, we have defined a set of custom metrics using TensorFlow's Keras metrics, including Categorical Accuracy, AUC and F1 Score. These metrics will be used to measure various aspects of the model's performance.

We have also defined a custom callback function to stop training our model when accuracy goes above 99%.

```
model = construct_model()
```

```
METRICS = [tf.keras.metrics.CategoricalAccuracy(name='acc'),tf.keras.metrics.AUC(name='auc'),tfa.metrics.F1Score(num_classes=4)]
model.compile(optimizer='adam',loss=tf.losses.CategoricalCrossentropy(),metrics=METRICS)
```

```
#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('val_acc') > 0.99:
            print("\nReached accuracy threshold! Terminating training.")
            self.model.stop_training = True

my_callback = MyCallback()
```

Activity 3: Train the Model

Now, we will train our model with our image dataset. The model is trained for 100 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch and training accuracy increases. The fit function is used to train a deep learning neural network.

```
#Fit the training data to the model and validate it using the validation data
model.fit(train_data, train_labels, validation_data=(val_data, val_labels),callbacks=my_callback,epochs=100)
```

```
Epoch 1/100
256/256 [==============================] - 145s 562ms/step - loss: 1.6891 - acc: 0.2765 - auc: 0.5282 - f1_score:
0.2753 - val_loss: 1.9973 - val_acc: 0.2422 - val_auc: 0.5185 - val_f1_score: 0.0975
Epoch 2/100
256/256 [==============================] - 145s 567ms/step - loss: 1.4286 - acc: 0.3192 - auc: 0.5847 - f1_score:
0.3145 - val_loss: 1.5013 - val_acc: 0.2393 - val_auc: 0.5012 - val_f1_score: 0.0987
Epoch 3/100
256/256 [==============================] - 816s 3s/step - loss: 0.9941 - acc: 0.5436 - auc: 0.8183 - f1_score: 0.52
02 - val_loss: 1.8700 - val_acc: 0.4146 - val_auc: 0.6506 - val_f1_score: 0.3658
Epoch 4/100
256/256 [==============================] - 153s 595ms/step - loss: 0.7979 - acc: 0.6385 - auc: 0.8830 - f1_score:
0.6302 - val_loss: 2.1478 - val_acc: 0.4775 - val_auc: 0.7073 - val_f1_score: 0.3594
Epoch 5/100
256/256 [==============================] - 155s 605ms/step - loss: 0.6883 - acc: 0.6904 - auc: 0.9125 - f1_score:
0.6841 - val_loss: 1.1839 - val_acc: 0.4956 - val_auc: 0.8323 - val_f1_score: 0.3769
Epoch 6/100
256/256 [==============================] - 233s 912ms/step - loss: 0.6539 - acc: 0.7031 - auc: 0.9213 - f1_score:
0.6997 - val_loss: 1.0413 - val_acc: 0.6060 - val_auc: 0.8701 - val_f1_score: 0.5719
```

```
Epoch 7/100
256/256 [==============================] - 216s 845ms/step - loss: 0.6184 - acc: 0.7233 - auc: 0.9297 - f1_score:
0.7227 - val_loss: 0.6159 - val_acc: 0.6958 - val_auc: 0.9264 - val_f1_score: 0.6964
Epoch 8/100
256/256 [==============================] - 226s 884ms/step - loss: 0.5485 - acc: 0.7570 - auc: 0.9445 - f1_score:
0.7564 - val_loss: 0.6865 - val_acc: 0.6328 - val_auc: 0.9048 - val_f1_score: 0.6020
Epoch 9/100
256/256 [==============================] - 251s 981ms/step - loss: 0.5481 - acc: 0.7615 - auc: 0.9447 - f1_score:
0.7612 - val_loss: 0.8633 - val_acc: 0.6333 - val_auc: 0.8720 - val_f1_score: 0.6429
Epoch 10/100
256/256 [==============================] - 223s 871ms/step - loss: 0.4852 - acc: 0.7883 - auc: 0.9562 - f1_score:
0.7888 - val_loss: 0.6715 - val_acc: 0.6777 - val_auc: 0.9150 - val_f1_score: 0.6657
Epoch 11/100
256/256 [==============================] - 213s 834ms/step - loss: 0.4496 - acc: 0.8059 - auc: 0.9627 - f1_score:
0.8060 - val_loss: 0.4677 - val_acc: 0.7920 - val_auc: 0.9592 - val_f1_score: 0.7865
Epoch 12/100
256/256 [==============================] - 219s 854ms/step - loss: 0.4074 - acc: 0.8285 - auc: 0.9689 - f1_score:
0.8288 - val_loss: 0.8827 - val_acc: 0.6479 - val_auc: 0.8891 - val_f1_score: 0.5976
```

```
Epoch 13/100
256/256 [==============================] - 218s 850ms/step - loss: 0.4339 - acc: 0.8164 - auc: 0.9654 - f1_score:
0.8166 - val_loss: 0.5375 - val_acc: 0.7305 - val_auc: 0.9532 - val_f1_score: 0.6629
Epoch 14/100
256/256 [==============================] - 219s 855ms/step - loss: 0.3637 - acc: 0.8467 - auc: 0.9752 - f1_score:
0.8462 - val_loss: 1.6647 - val_acc: 0.5508 - val_auc: 0.8326 - val_f1_score: 0.4944
Epoch 15/100
256/256 [==============================] - 213s 832ms/step - loss: 0.3477 - acc: 0.8601 - auc: 0.9774 - f1_score:
0.8601 - val_loss: 1.2978 - val_acc: 0.5728 - val_auc: 0.8516 - val_f1_score: 0.5450
Epoch 16/100
256/256 [==============================] - 214s 836ms/step - loss: 0.3018 - acc: 0.8799 - auc: 0.9827 - f1_score:
0.8797 - val_loss: 0.4002 - val_acc: 0.8330 - val_auc: 0.9729 - val_f1_score: 0.8344
Epoch 17/100
256/256 [==============================] - 233s 910ms/step - loss: 0.2894 - acc: 0.8827 - auc: 0.9841 - f1_score:
0.8825 - val_loss: 0.6789 - val_acc: 0.7505 - val_auc: 0.9537 - val_f1_score: 0.7156
Epoch 18/100
256/256 [==============================] - 246s 962ms/step - loss: 0.2828 - acc: 0.8903 - auc: 0.9848 - f1_score:
0.8901 - val_loss: 0.3758 - val_acc: 0.8521 - val_auc: 0.9766 - val_f1_score: 0.8523
```

```
Epoch 19/100
256/256 [==============================] - 252s 986ms/step - loss: 0.2316 - acc: 0.9147 - auc: 0.9893 - f1_score:
0.9145 - val_loss: 0.5368 - val_acc: 0.7900 - val_auc: 0.9605 - val_f1_score: 0.7639
Epoch 20/100
256/256 [==============================] - 241s 941ms/step - loss: 0.1983 - acc: 0.9264 - auc: 0.9922 - f1_score:
0.9264 - val_loss: 0.8691 - val_acc: 0.7163 - val_auc: 0.9227 - val_f1_score: 0.7108
Epoch 21/100
256/256 [==============================] - 240s 939ms/step - loss: 0.1885 - acc: 0.9327 - auc: 0.9928 - f1_score:
0.9326 - val_loss: 0.6804 - val_acc: 0.7798 - val_auc: 0.9473 - val_f1_score: 0.7735
Epoch 22/100
256/256 [==============================] - 239s 934ms/step - loss: 0.1591 - acc: 0.9409 - auc: 0.9947 - f1_score:
0.9407 - val_loss: 0.2856 - val_acc: 0.8896 - val_auc: 0.9861 - val_f1_score: 0.8877
Epoch 23/100
256/256 [==============================] - 248s 967ms/step - loss: 0.1460 - acc: 0.9482 - auc: 0.9951 - f1_score:
0.9482 - val_loss: 1.2427 - val_acc: 0.6699 - val_auc: 0.8628 - val_f1_score: 0.6741
Epoch 24/100
256/256 [==============================] - 243s 948ms/step - loss: 0.1459 - acc: 0.9476 - auc: 0.9951 - f1_score:
0.9476 - val_loss: 0.4146 - val_acc: 0.8491 - val_auc: 0.9790 - val_f1_score: 0.8425


Epoch 25/100
256/256 [==============================] - 292s 1s/step - loss: 0.1275 - acc: 0.9585 - auc: 0.9959 - f1_score: 0.95
84 - val_loss: 0.1938 - val_acc: 0.9277 - val_auc: 0.9928 - val_f1_score: 0.9274
Epoch 26/100
256/256 [==============================] - 273s 1s/step - loss: 0.1140 - acc: 0.9590 - auc: 0.9972 - f1_score: 0.95
89 - val_loss: 0.3242 - val_acc: 0.8940 - val_auc: 0.9834 - val_f1_score: 0.8938
Epoch 27/100
256/256 [==============================] - 251s 980ms/step - loss: 0.1071 - acc: 0.9626 - auc: 0.9971 - f1_score:
0.9626 - val_loss: 0.2522 - val_acc: 0.9077 - val_auc: 0.9902 - val_f1_score: 0.9056
Epoch 28/100
256/256 [==============================] - 233s 909ms/step - loss: 0.0856 - acc: 0.9739 - auc: 0.9978 - f1_score:
0.9739 - val_loss: 0.3217 - val_acc: 0.8882 - val_auc: 0.9831 - val_f1_score: 0.8898
Epoch 29/100
256/256 [==============================] - 235s 918ms/step - loss: 0.0934 - acc: 0.9691 - auc: 0.9975 - f1_score:
0.9691 - val_loss: 0.3955 - val_acc: 0.8828 - val_auc: 0.9802 - val_f1_score: 0.8757
Epoch 30/100
256/256 [==============================] - 247s 965ms/step - loss: 0.0907 - acc: 0.9697 - auc: 0.9975 - f1_score:
0.9697 - val_loss: 0.3866 - val_acc: 0.8745 - val_auc: 0.9816 - val_f1_score: 0.8702


Epoch 31/100
256/256 [==============================] - 237s 926ms/step - loss: 0.0759 - acc: 0.9764 - auc: 0.9983 - f1_score:
0.9764 - val_loss: 0.2729 - val_acc: 0.9116 - val_auc: 0.9872 - val_f1_score: 0.9101
Epoch 32/100
256/256 [==============================] - 211s 825ms/step - loss: 0.0983 - acc: 0.9657 - auc: 0.9976 - f1_score:
0.9657 - val_loss: 0.2741 - val_acc: 0.9043 - val_auc: 0.9880 - val_f1_score: 0.9059
Epoch 33/100
256/256 [==============================] - 219s 855ms/step - loss: 0.0694 - acc: 0.9766 - auc: 0.9988 - f1_score:
0.9766 - val_loss: 0.2249 - val_acc: 0.9268 - val_auc: 0.9907 - val_f1_score: 0.9258
Epoch 34/100
256/256 [==============================] - 213s 833ms/step - loss: 0.0746 - acc: 0.9761 - auc: 0.9979 - f1_score:
0.9761 - val_loss: 0.5122 - val_acc: 0.8633 - val_auc: 0.9691 - val_f1_score: 0.8593
Epoch 35/100
256/256 [==============================] - 213s 832ms/step - loss: 0.0665 - acc: 0.9795 - auc: 0.9982 - f1_score:
0.9795 - val_loss: 0.2069 - val_acc: 0.9331 - val_auc: 0.9917 - val_f1_score: 0.9327
Epoch 36/100
256/256 [==============================] - 212s 826ms/step - loss: 0.0673 - acc: 0.9797 - auc: 0.9982 - f1_score:
0.9797 - val_loss: 0.3220 - val_acc: 0.9004 - val_auc: 0.9843 - val_f1_score: 0.8989


Epoch 37/100
256/256 [==============================] - 213s 832ms/step - loss: 0.0702 - acc: 0.9773 - auc: 0.9985 - f1_score:
0.9773 - val_loss: 0.2536 - val_acc: 0.9238 - val_auc: 0.9869 - val_f1_score: 0.9242
Epoch 38/100
256/256 [==============================] - 218s 853ms/step - loss: 0.0454 - acc: 0.9861 - auc: 0.9993 - f1_score:
0.9861 - val_loss: 0.2527 - val_acc: 0.9331 - val_auc: 0.9845 - val_f1_score: 0.9324
Epoch 39/100
256/256 [==============================] - 214s 836ms/step - loss: 0.0564 - acc: 0.9816 - auc: 0.9986 - f1_score:
0.9816 - val_loss: 0.3924 - val_acc: 0.8921 - val_auc: 0.9762 - val_f1_score: 0.8933

Epoch 40/100
256/256 [==============================] - 207s 810ms/step - loss: 0.0514 - acc: 0.9829 - auc: 0.9990 - f1_score:
0.9829 - val_loss: 1.1440 - val_acc: 0.7554 - val_auc: 0.9148 - val_f1_score: 0.7481
Epoch 41/100
256/256 [==============================] - 203s 792ms/step - loss: 0.0538 - acc: 0.9832 - auc: 0.9990 - f1_score:
0.9832 - val_loss: 0.2511 - val_acc: 0.9253 - val_auc: 0.9899 - val_f1_score: 0.9243
Epoch 42/100
256/256 [==============================] - 215s 840ms/step - loss: 0.0508 - acc: 0.9839 - auc: 0.9991 - f1_score:
0.9839 - val_loss: 0.5120 - val_acc: 0.8608 - val_auc: 0.9717 - val_f1_score: 0.8501
```

```
Epoch 43/100
256/256 [==============================] - 201s 784ms/step - loss: 0.0578 - acc: 0.9822 - auc: 0.9986 - f1_score:
0.9822 - val_loss: 0.2956 - val_acc: 0.9048 - val_auc: 0.9865 - val_f1_score: 0.9033
Epoch 44/100
256/256 [==============================] - 200s 781ms/step - loss: 0.0605 - acc: 0.9799 - auc: 0.9983 - f1_score:
0.9798 - val_loss: 0.1963 - val_acc: 0.9390 - val_auc: 0.9920 - val_f1_score: 0.9388
Epoch 45/100
256/256 [==============================] - 203s 793ms/step - loss: 0.0465 - acc: 0.9854 - auc: 0.9992 - f1_score:
0.9853 - val_loss: 0.1754 - val_acc: 0.9512 - val_auc: 0.9921 - val_f1_score: 0.9507
Epoch 46/100
256/256 [==============================] - 204s 797ms/step - loss: 0.0302 - acc: 0.9910 - auc: 0.9996 - f1_score:
0.9910 - val_loss: 0.6197 - val_acc: 0.8555 - val_auc: 0.9593 - val_f1_score: 0.8549
Epoch 47/100
256/256 [==============================] - 197s 769ms/step - loss: 0.0794 - acc: 0.9725 - auc: 0.9979 - f1_score:
0.9725 - val_loss: 0.2373 - val_acc: 0.9297 - val_auc: 0.9894 - val_f1_score: 0.9285
Epoch 48/100
256/256 [==============================] - 195s 761ms/step - loss: 0.0374 - acc: 0.9879 - auc: 0.9995 - f1_score:
0.9879 - val_loss: 0.2649 - val_acc: 0.9292 - val_auc: 0.9846 - val_f1_score: 0.9293

Epoch 49/100
256/256 [==============================] - 196s 766ms/step - loss: 0.0338 - acc: 0.9890 - auc: 0.9993 - f1_score:
0.9890 - val_loss: 0.1791 - val_acc: 0.9473 - val_auc: 0.9923 - val_f1_score: 0.9468
Epoch 50/100
256/256 [==============================] - 209s 815ms/step - loss: 0.0491 - acc: 0.9847 - auc: 0.9989 - f1_score:
0.9847 - val_loss: 0.1678 - val_acc: 0.9521 - val_auc: 0.9918 - val_f1_score: 0.9522
Epoch 51/100
256/256 [==============================] - 179s 700ms/step - loss: 0.0329 - acc: 0.9907 - auc: 0.9995 - f1_score:
0.9907 - val_loss: 0.2764 - val_acc: 0.9258 - val_auc: 0.9859 - val_f1_score: 0.9241
Epoch 52/100
256/256 [==============================] - 179s 700ms/step - loss: 0.0335 - acc: 0.9894 - auc: 0.9995 - f1_score:
0.9894 - val_loss: 0.2164 - val_acc: 0.9414 - val_auc: 0.9895 - val_f1_score: 0.9406
Epoch 53/100
256/256 [==============================] - 187s 729ms/step - loss: 0.0410 - acc: 0.9869 - auc: 0.9993 - f1_score:
0.9869 - val_loss: 0.5101 - val_acc: 0.8667 - val_auc: 0.9631 - val_f1_score: 0.8676
Epoch 54/100
256/256 [==============================] - 193s 755ms/step - loss: 0.0475 - acc: 0.9856 - auc: 0.9988 - f1_score:
0.9856 - val_loss: 0.3262 - val_acc: 0.9165 - val_auc: 0.9787 - val_f1_score: 0.9156

Epoch 55/100
256/256 [==============================] - 195s 761ms/step - loss: 0.0360 - acc: 0.9882 - auc: 0.9995 - f1_score:
0.9882 - val_loss: 0.2790 - val_acc: 0.9263 - val_auc: 0.9854 - val_f1_score: 0.9256
Epoch 56/100
256/256 [==============================] - 191s 747ms/step - loss: 0.0228 - acc: 0.9938 - auc: 0.9997 - f1_score:
0.9938 - val_loss: 0.3429 - val_acc: 0.9233 - val_auc: 0.9771 - val_f1_score: 0.9233
Epoch 57/100
256/256 [==============================] - 190s 742ms/step - loss: 0.0303 - acc: 0.9899 - auc: 0.9994 - f1_score:
0.9899 - val_loss: 0.2657 - val_acc: 0.9307 - val_auc: 0.9845 - val_f1_score: 0.9297
Epoch 58/100
256/256 [==============================] - 182s 710ms/step - loss: 0.0430 - acc: 0.9862 - auc: 0.9990 - f1_score:
0.9862 - val_loss: 0.1925 - val_acc: 0.9458 - val_auc: 0.9917 - val_f1_score: 0.9459
Epoch 59/100
256/256 [==============================] - 183s 715ms/step - loss: 0.0351 - acc: 0.9888 - auc: 0.9994 - f1_score:
0.9888 - val_loss: 0.8623 - val_acc: 0.7896 - val_auc: 0.9312 - val_f1_score: 0.7827
Epoch 60/100
256/256 [==============================] - 183s 715ms/step - loss: 0.0373 - acc: 0.9880 - auc: 0.9994 - f1_score:
0.9880 - val_loss: 0.2484 - val_acc: 0.9331 - val_auc: 0.9877 - val_f1_score: 0.9322

Epoch 61/100
256/256 [==============================] - 184s 719ms/step - loss: 0.0360 - acc: 0.9901 - auc: 0.9991 - f1_score:
0.9901 - val_loss: 0.1849 - val_acc: 0.9507 - val_auc: 0.9916 - val_f1_score: 0.9501
Epoch 62/100
256/256 [==============================] - 197s 770ms/step - loss: 0.0193 - acc: 0.9941 - auc: 0.9997 - f1_score:
0.9941 - val_loss: 0.4036 - val_acc: 0.9077 - val_auc: 0.9757 - val_f1_score: 0.9070
Epoch 63/100
256/256 [==============================] - 201s 785ms/step - loss: 0.0340 - acc: 0.9901 - auc: 0.9992 - f1_score:
0.9901 - val_loss: 0.2027 - val_acc: 0.9448 - val_auc: 0.9896 - val_f1_score: 0.9450
Epoch 64/100
256/256 [==============================] - 200s 783ms/step - loss: 0.0439 - acc: 0.9861 - auc: 0.9990 - f1_score:
0.9861 - val_loss: 0.1760 - val_acc: 0.9521 - val_auc: 0.9913 - val_f1_score: 0.9522
Epoch 65/100
256/256 [==============================] - 201s 784ms/step - loss: 0.0369 - acc: 0.9897 - auc: 0.9994 - f1_score:
0.9897 - val_loss: 0.1864 - val_acc: 0.9473 - val_auc: 0.9909 - val_f1_score: 0.9472
Epoch 66/100
256/256 [==============================] - 203s 792ms/step - loss: 0.0375 - acc: 0.9877 - auc: 0.9990 - f1_score:
0.9877 - val_loss: 0.1647 - val_acc: 0.9541 - val_auc: 0.9925 - val_f1_score: 0.9539
```

```
Epoch 67/100
256/256 [==============================] - 202s 789ms/step - loss: 0.0201 - acc: 0.9940 - auc: 0.9997 - f1_score:
0.9940 - val_loss: 0.2174 - val_acc: 0.9453 - val_auc: 0.9893 - val_f1_score: 0.9452
Epoch 68/100
256/256 [==============================] - 204s 797ms/step - loss: 0.0295 - acc: 0.9911 - auc: 0.9994 - f1_score:
0.9911 - val_loss: 0.1949 - val_acc: 0.9492 - val_auc: 0.9909 - val_f1_score: 0.9489
Epoch 69/100
256/256 [==============================] - 204s 797ms/step - loss: 0.0307 - acc: 0.9908 - auc: 0.9994 - f1_score:
0.9908 - val_loss: 0.1685 - val_acc: 0.9507 - val_auc: 0.9924 - val_f1_score: 0.9504
Epoch 70/100
256/256 [==============================] - 204s 798ms/step - loss: 0.0330 - acc: 0.9891 - auc: 0.9991 - f1_score:
0.9891 - val_loss: 0.1688 - val_acc: 0.9536 - val_auc: 0.9922 - val_f1_score: 0.9538
Epoch 71/100
256/256 [==============================] - 205s 802ms/step - loss: 0.0260 - acc: 0.9927 - auc: 0.9995 - f1_score:
0.9927 - val_loss: 0.1847 - val_acc: 0.9507 - val_auc: 0.9917 - val_f1_score: 0.9502
Epoch 72/100
256/256 [==============================] - 204s 799ms/step - loss: 0.0182 - acc: 0.9946 - auc: 0.9996 - f1_score:
0.9946 - val_loss: 0.2271 - val_acc: 0.9443 - val_auc: 0.9869 - val_f1_score: 0.9447

Epoch 73/100
256/256 [==============================] - 205s 800ms/step - loss: 0.0256 - acc: 0.9932 - auc: 0.9993 - f1_score:
0.9932 - val_loss: 0.1973 - val_acc: 0.9463 - val_auc: 0.9900 - val_f1_score: 0.9459
Epoch 74/100
256/256 [==============================] - 206s 803ms/step - loss: 0.0250 - acc: 0.9919 - auc: 0.9996 - f1_score:
0.9919 - val_loss: 0.1898 - val_acc: 0.9497 - val_auc: 0.9896 - val_f1_score: 0.9499
Epoch 75/100
256/256 [==============================] - 206s 805ms/step - loss: 0.0200 - acc: 0.9930 - auc: 0.9998 - f1_score:
0.9930 - val_loss: 0.2569 - val_acc: 0.9419 - val_auc: 0.9854 - val_f1_score: 0.9422
Epoch 76/100
256/256 [==============================] - 207s 808ms/step - loss: 0.0175 - acc: 0.9950 - auc: 0.9996 - f1_score:
0.9950 - val_loss: 0.3128 - val_acc: 0.9287 - val_auc: 0.9804 - val_f1_score: 0.9287
Epoch 77/100
256/256 [==============================] - 207s 810ms/step - loss: 0.0396 - acc: 0.9886 - auc: 0.9987 - f1_score:
0.9886 - val_loss: 0.1571 - val_acc: 0.9541 - val_auc: 0.9944 - val_f1_score: 0.9538
Epoch 78/100
256/256 [==============================] - 208s 813ms/step - loss: 0.0163 - acc: 0.9957 - auc: 0.9997 - f1_score:
0.9957 - val_loss: 0.2138 - val_acc: 0.9482 - val_auc: 0.9885 - val_f1_score: 0.9479

Epoch 79/100
256/256 [==============================] - 206s 806ms/step - loss: 0.0170 - acc: 0.9946 - auc: 0.9997 - f1_score:
0.9946 - val_loss: 0.2035 - val_acc: 0.9507 - val_auc: 0.9880 - val_f1_score: 0.9506
Epoch 80/100
256/256 [==============================] - 205s 802ms/step - loss: 0.0230 - acc: 0.9937 - auc: 0.9996 - f1_score:
0.9937 - val_loss: 0.2214 - val_acc: 0.9414 - val_auc: 0.9883 - val_f1_score: 0.9405
Epoch 81/100
256/256 [==============================] - 208s 811ms/step - loss: 0.0277 - acc: 0.9922 - auc: 0.9994 - f1_score:
0.9922 - val_loss: 0.2643 - val_acc: 0.9287 - val_auc: 0.9857 - val_f1_score: 0.9292
Epoch 82/100
256/256 [==============================] - 205s 802ms/step - loss: 0.0252 - acc: 0.9918 - auc: 0.9994 - f1_score:
0.9918 - val_loss: 0.2049 - val_acc: 0.9507 - val_auc: 0.9875 - val_f1_score: 0.9509
Epoch 83/100
256/256 [==============================] - 205s 800ms/step - loss: 0.0206 - acc: 0.9938 - auc: 0.9998 - f1_score:
0.9938 - val_loss: 0.2573 - val_acc: 0.9370 - val_auc: 0.9828 - val_f1_score: 0.9374
Epoch 84/100
256/256 [==============================] - 205s 801ms/step - loss: 0.0151 - acc: 0.9956 - auc: 0.9998 - f1_score:
0.9956 - val_loss: 0.1761 - val_acc: 0.9526 - val_auc: 0.9912 - val_f1_score: 0.9525

Epoch 85/100
256/256 [==============================] - 206s 804ms/step - loss: 0.0232 - acc: 0.9933 - auc: 0.9995 - f1_score:
0.9933 - val_loss: 0.1840 - val_acc: 0.9502 - val_auc: 0.9918 - val_f1_score: 0.9501
Epoch 86/100
256/256 [==============================] - 211s 826ms/step - loss: 0.0170 - acc: 0.9944 - auc: 0.9997 - f1_score:
0.9944 - val_loss: 0.2619 - val_acc: 0.9443 - val_auc: 0.9863 - val_f1_score: 0.9441
Epoch 87/100
256/256 [==============================] - 207s 808ms/step - loss: 0.0157 - acc: 0.9958 - auc: 0.9996 - f1_score:
0.9958 - val_loss: 0.2902 - val_acc: 0.9365 - val_auc: 0.9817 - val_f1_score: 0.9361
Epoch 88/100
256/256 [==============================] - 206s 806ms/step - loss: 0.0236 - acc: 0.9927 - auc: 0.9996 - f1_score:
0.9927 - val_loss: 0.1584 - val_acc: 0.9590 - val_auc: 0.9924 - val_f1_score: 0.9589
Epoch 89/100
256/256 [==============================] - 205s 801ms/step - loss: 0.0222 - acc: 0.9932 - auc: 0.9996 - f1_score:
0.9932 - val_loss: 1.0883 - val_acc: 0.7988 - val_auc: 0.9231 - val_f1_score: 0.7939
Epoch 90/100
256/256 [==============================] - 204s 798ms/step - loss: 0.0312 - acc: 0.9904 - auc: 0.9994 - f1_score:
0.9904 - val_loss: 0.1729 - val_acc: 0.9561 - val_auc: 0.9897 - val_f1_score: 0.9562
```

```
Epoch 91/100
256/256 [==============================] - 205s 801ms/step - loss: 0.0188 - acc: 0.9943 - auc: 0.9996 - f1_score:
0.9943 - val_loss: 0.1855 - val_acc: 0.9541 - val_auc: 0.9916 - val_f1_score: 0.9540
Epoch 92/100
256/256 [==============================] - 209s 816ms/step - loss: 0.0124 - acc: 0.9963 - auc: 0.9999 - f1_score:
0.9963 - val_loss: 0.2140 - val_acc: 0.9521 - val_auc: 0.9888 - val_f1_score: 0.9523
Epoch 93/100
256/256 [==============================] - 206s 807ms/step - loss: 0.0123 - acc: 0.9973 - auc: 0.9997 - f1_score:
0.9973 - val_loss: 0.3036 - val_acc: 0.9375 - val_auc: 0.9874 - val_f1_score: 0.9367
Epoch 94/100
256/256 [==============================] - 208s 812ms/step - loss: 0.0185 - acc: 0.9944 - auc: 0.9996 - f1_score:
0.9944 - val_loss: 0.3299 - val_acc: 0.9375 - val_auc: 0.9854 - val_f1_score: 0.9366
Epoch 95/100
256/256 [==============================] - 210s 820ms/step - loss: 0.0392 - acc: 0.9884 - auc: 0.9988 - f1_score:
0.9884 - val_loss: 0.1867 - val_acc: 0.9497 - val_auc: 0.9911 - val_f1_score: 0.9492
Epoch 96/100
256/256 [==============================] - 215s 839ms/step - loss: 0.0120 - acc: 0.9965 - auc: 0.9999 - f1_score:
0.9965 - val_loss: 0.2038 - val_acc: 0.9521 - val_auc: 0.9881 - val_f1_score: 0.9522

Epoch 97/100
256/256 [==============================] - 212s 830ms/step - loss: 0.0150 - acc: 0.9957 - auc: 0.9997 - f1_score:
0.9957 - val_loss: 0.2031 - val_acc: 0.9521 - val_auc: 0.9885 - val_f1_score: 0.9518
Epoch 98/100
256/256 [==============================] - 205s 802ms/step - loss: 0.0188 - acc: 0.9945 - auc: 0.9996 - f1_score:
0.9945 - val_loss: 0.1918 - val_acc: 0.9546 - val_auc: 0.9887 - val_f1_score: 0.9546
Epoch 99/100
256/256 [==============================] - 204s 795ms/step - loss: 0.0188 - acc: 0.9943 - auc: 0.9998 - f1_score:
0.9943 - val_loss: 0.2093 - val_acc: 0.9521 - val_auc: 0.9873 - val_f1_score: 0.9516
Epoch 100/100
256/256 [==============================] - 204s 796ms/step - loss: 0.0089 - acc: 0.9972 - auc: 1.0000 - f1_score:
0.9972 - val_loss: 0.2315 - val_acc: 0.9551 - val_auc: 0.9872 - val_f1_score: 0.9548

<keras.src.callbacks.History at 0x28a0286d0>
```

```
model.summary()
```

```
Model: "cnn_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 176, 176, 16)      448

 conv2d_1 (Conv2D)           (None, 176, 176, 16)      2320

 max_pooling2d (MaxPooling2  (None, 88, 88, 16)        0
 D)

 sequential (Sequential)     (None, 44, 44, 32)        14016

 sequential_1 (Sequential)   (None, 22, 22, 64)        55680

 sequential_2 (Sequential)   (None, 11, 11, 128)       221952

 dropout (Dropout)           (None, 11, 11, 128)       0
```

```
 sequential_3 (Sequential)   (None, 5, 5, 256)        886272

 dropout_1 (Dropout)         (None, 5, 5, 256)        0

 flatten (Flatten)           (None, 6400)             0

 sequential_4 (Sequential)   (None, 512)              3279360

 sequential_5 (Sequential)   (None, 128)              66176

 sequential_6 (Sequential)   (None, 64)               8512

 dense_3 (Dense)             (None, 4)                260

=================================================================
Total params: 4534996 (17.30 MB)
Trainable params: 4532628 (17.29 MB)
Non-trainable params: 2368 (9.25 KB)
_________________________________________________________________
```

Activity 4: Evaluate the Model

```
# Evaluating the model on the data

train_scores = model.evaluate(train_data, train_labels)
val_scores = model.evaluate(val_data, val_labels)
test_scores = model.evaluate(test_data, test_labels)

print("Training Accuracy: %.2f%%"%(train_scores[1] * 100))
print("Validation Accuracy: %.2f%%"%(val_scores[1] * 100))
print("Testing Accuracy: %.2f%%"%(test_scores[1] * 100))
```

```
256/256 [==============================] - 52s 203ms/step - loss: 0.0019 - acc: 0.9993 - auc: 1.0000 - f1_score: 0.
9993
64/64 [==============================] - 13s 200ms/step - loss: 0.2315 - acc: 0.9551 - auc: 0.9872 - f1_score: 0.95
48
80/80 [==============================] - 16s 200ms/step - loss: 0.2358 - acc: 0.9484 - auc: 0.9868 - f1_score: 0.94
82
Training Accuracy: 99.93%
Validation Accuracy: 95.51%
Testing Accuracy: 94.84%
```

Activity 5: Save the Model

```
#save model
model.save('adp.h5')
```

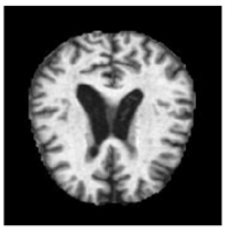
The model is saved with .h5 extension. A .h5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data

Activity 6: Testing the Model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Use an image to test the model and augment the image.

```
model = load_model(r"C:\Users\chris\Downloads\ADP Project\adp.h5",compile=False)
```

```
img = image.load_img(r"C:\Users\chris\Downloads\ADP Project\Dataset\ModerateDemented\moderateDem49.jpg",target_size=(176,176))
img
```



```
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
```

```
pred = model.predict(x)
pred_class = np.argmax(pred,axis=1)
index = ['MildDemented','ModerateDemented','NonDemented','VeryMildDemented']
result = str(index[pred_class[0]])
result
```

```
1/1 [==============================] - 0s 253ms/step

'ModerateDemented'
```

So, our model will give the index position of the label. In this case, the 1st index is for Moderate Demented, which has been predicted correctly.

Milestone 4: Application Building

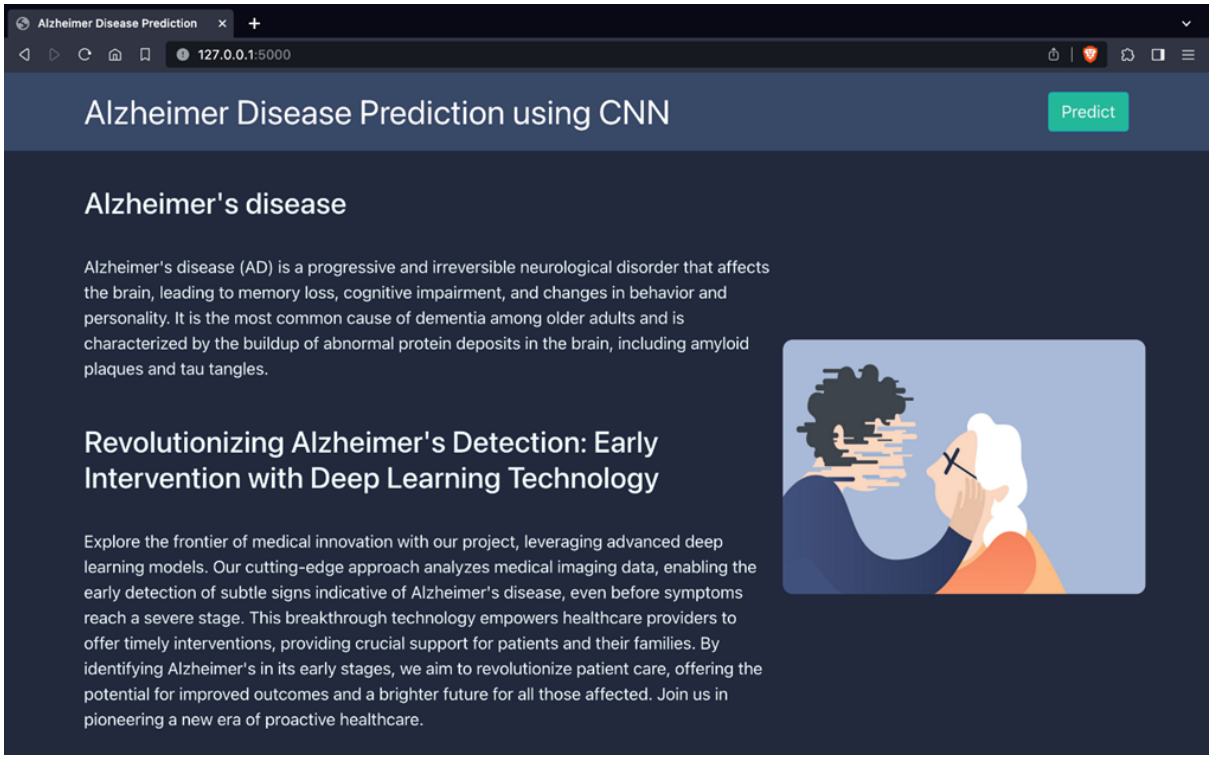
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

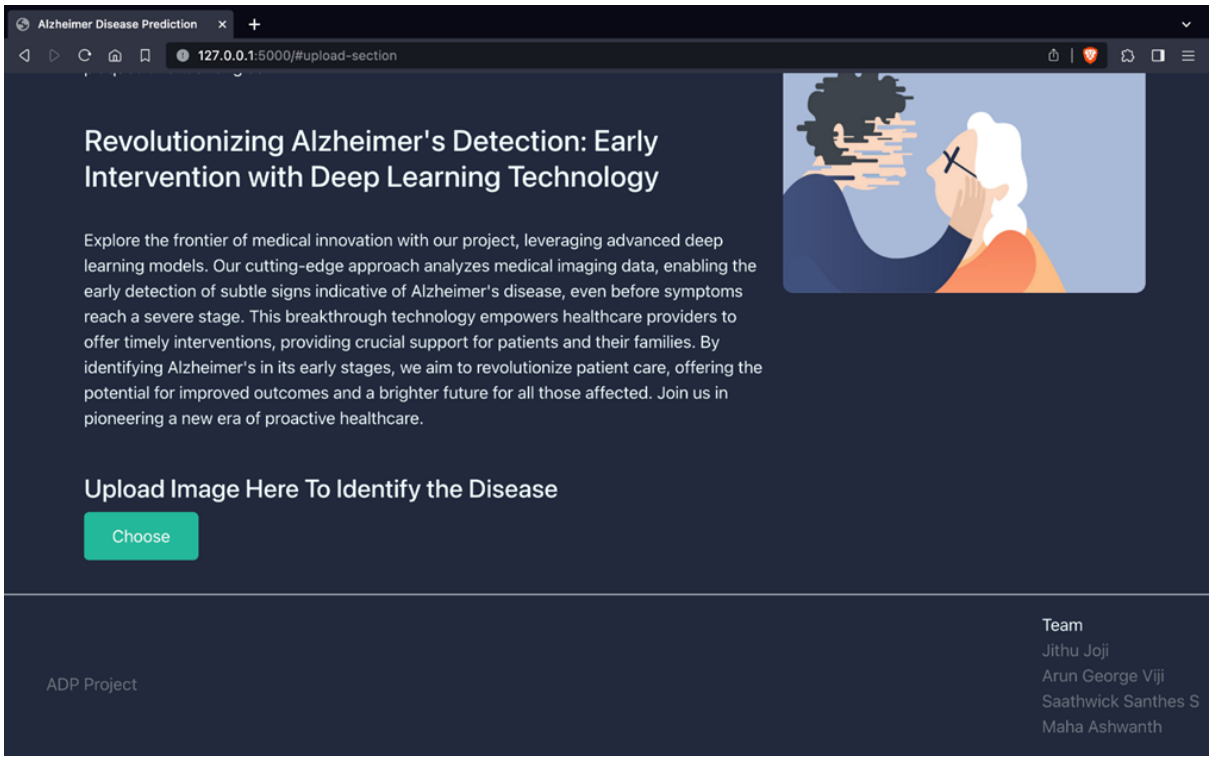
- Building HTML Pages
- Building python code

Activity 1: Building HTML Pages

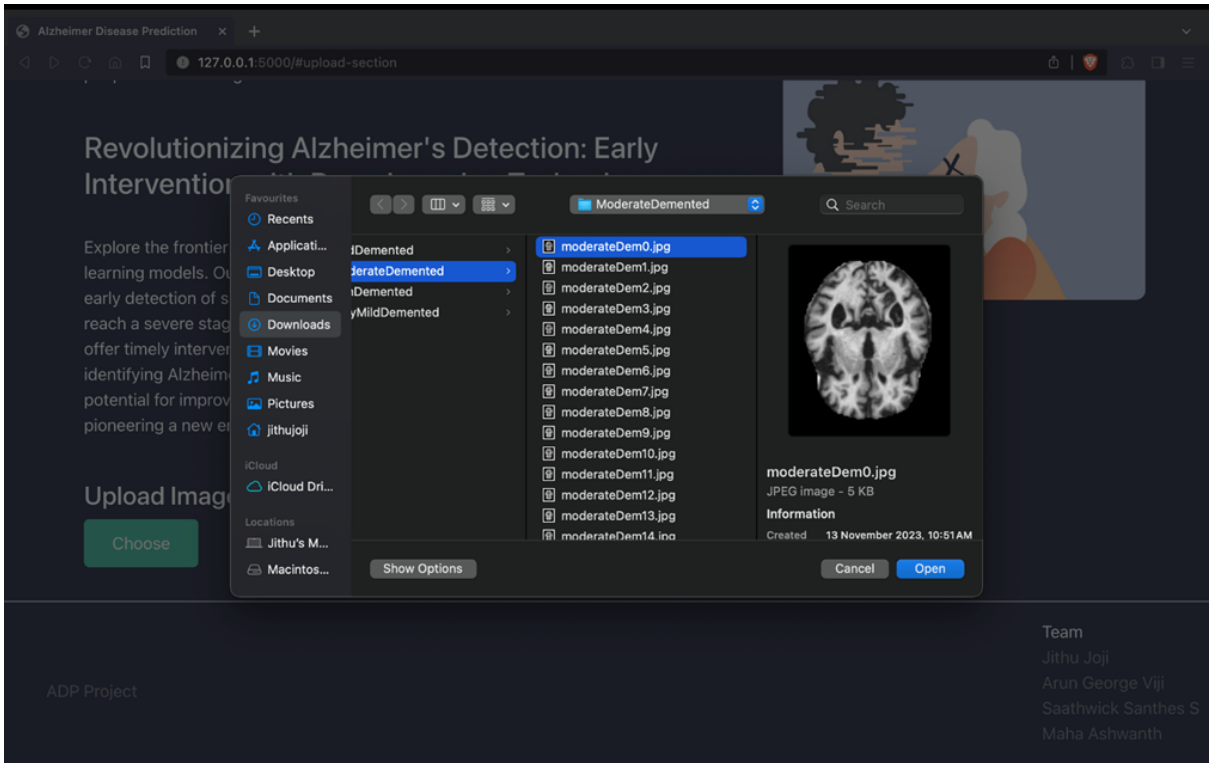
For this project create one HTML file, index.html. The index.html page looks as given below



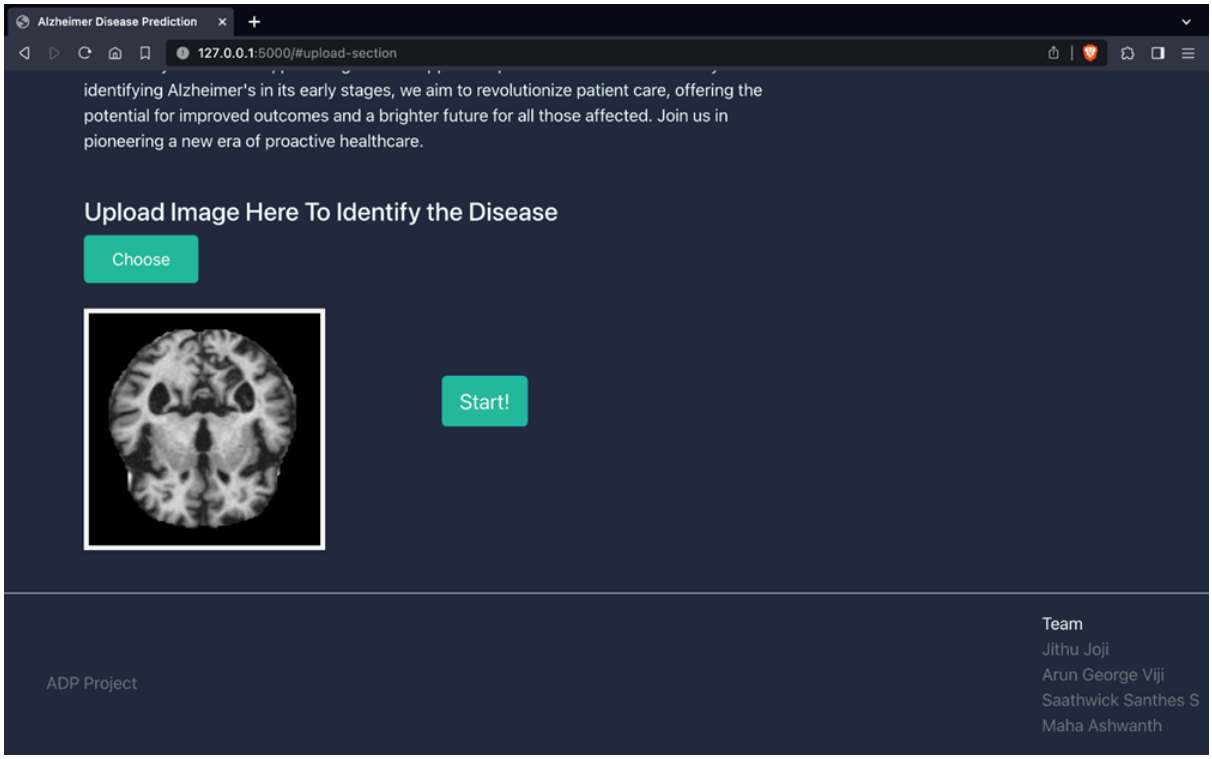
There is a predict button to redirect us to the upload image section.



When you click on the Choose button, it will redirect you to the below page



When you select the image, it will display as given below



Activity 2: Build Python code

Import the libraries

```
import numpy as np
import os
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask , request, render_template
```

Initializing Flask App and loading the saved model

```
app = Flask(__name__)

model = load_model("adp.h5",compile=False)
```

Render HTML Page

```
@app.route('/')
def index():
    return render_template('index.html')
```

Once we uploaded the file into the app, then verifying the file uploaded properly or not. Here we will be using a declared constructor to route to the HTML page which we have created earlier. Whenever you enter the values from the html page, the values can be retrieved using the POST Method.

```
@app.route('/predict',methods = ['GET','POST'])
def upload():
    if request.method == 'POST':
        f = request.files['image']
        print("current path")
        basepath = os.path.dirname(__file__)
        print("current path", basepath)
        filepath = os.path.join(basepath,'uploads',f.filename)
        print("upload folder is ", filepath)
        f.save(filepath)

        img = image.load_img(filepath,target_size = (176,176))
        x = image.img_to_array(img)
        print(x)
        x = np.expand_dims(x,axis =0)
        print(x)
        y=model.predict(x)
        preds=np.argmax(y, axis=1)
        print("Prediction",preds)
        index = ['Mild Demented','Moderate Demented','Non Demented','Very Mild Demented']
        text = "The person is " + str(index[preds[0]])
    return text
```

Here we are routing our app to upload function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the alzheimers.html page earlier.

Main Function

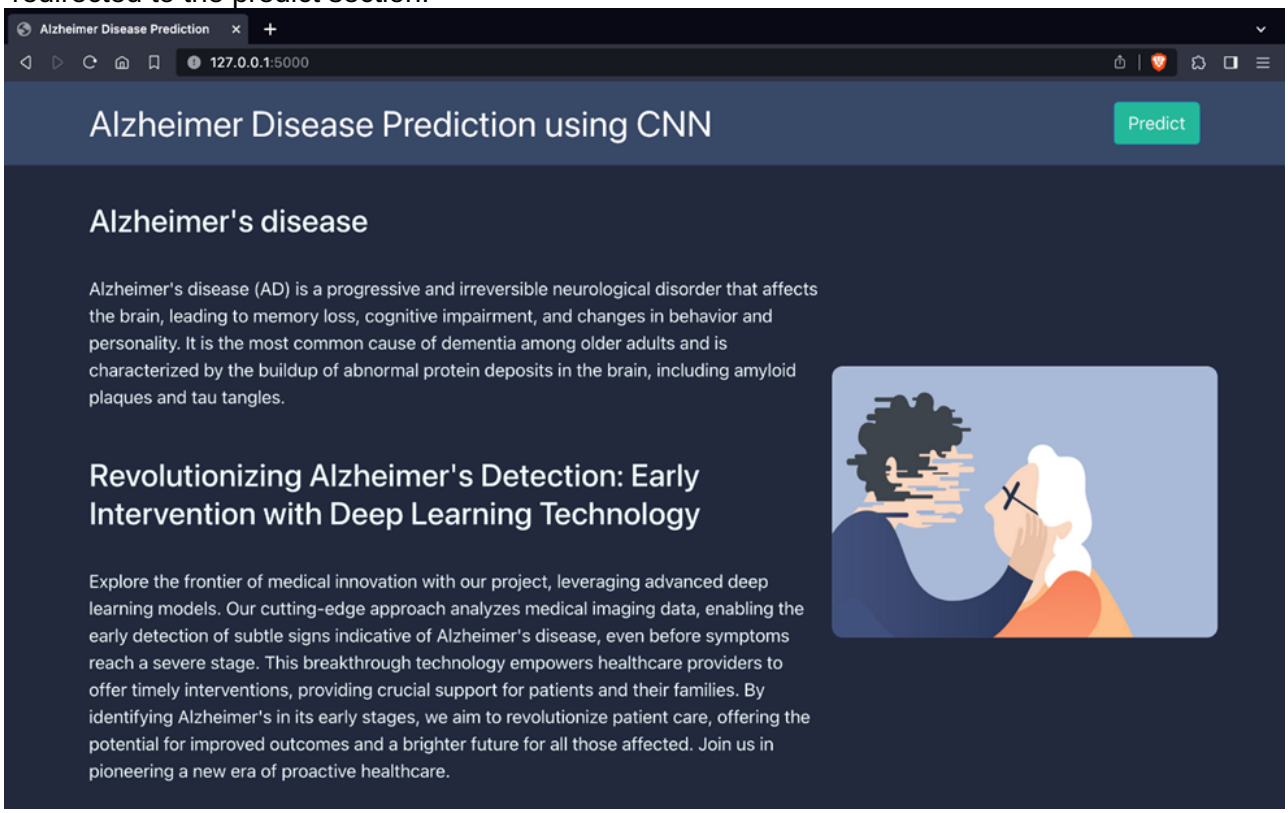
```
if __name__ == '__main__':
    app.run(debug = False, threaded = False)
```

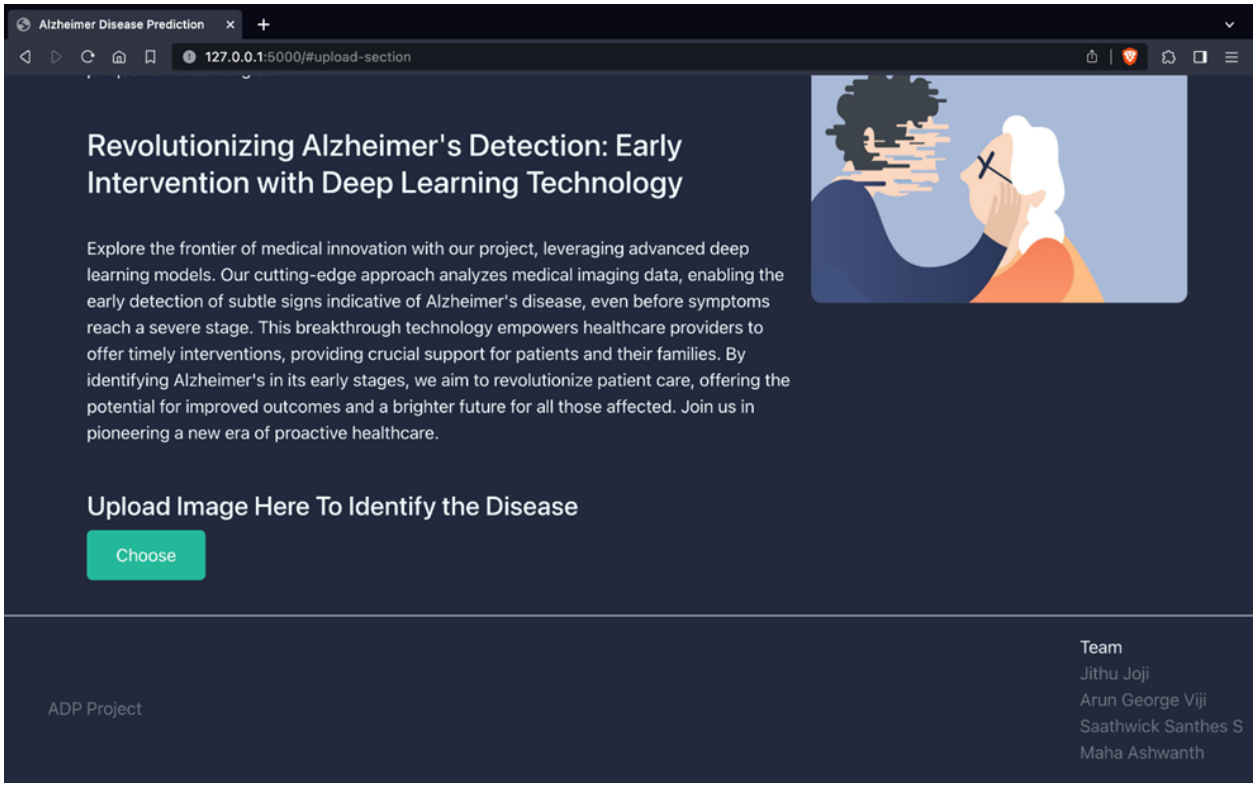
Activity 3: Run the Application

- Open Spyder
- Navigate to the folder where your Python script is.
- Now click on the Run button above.
- Click on the predict button from the top right corner, enter the inputs, click on the Start button, and see the result/prediction on the web.

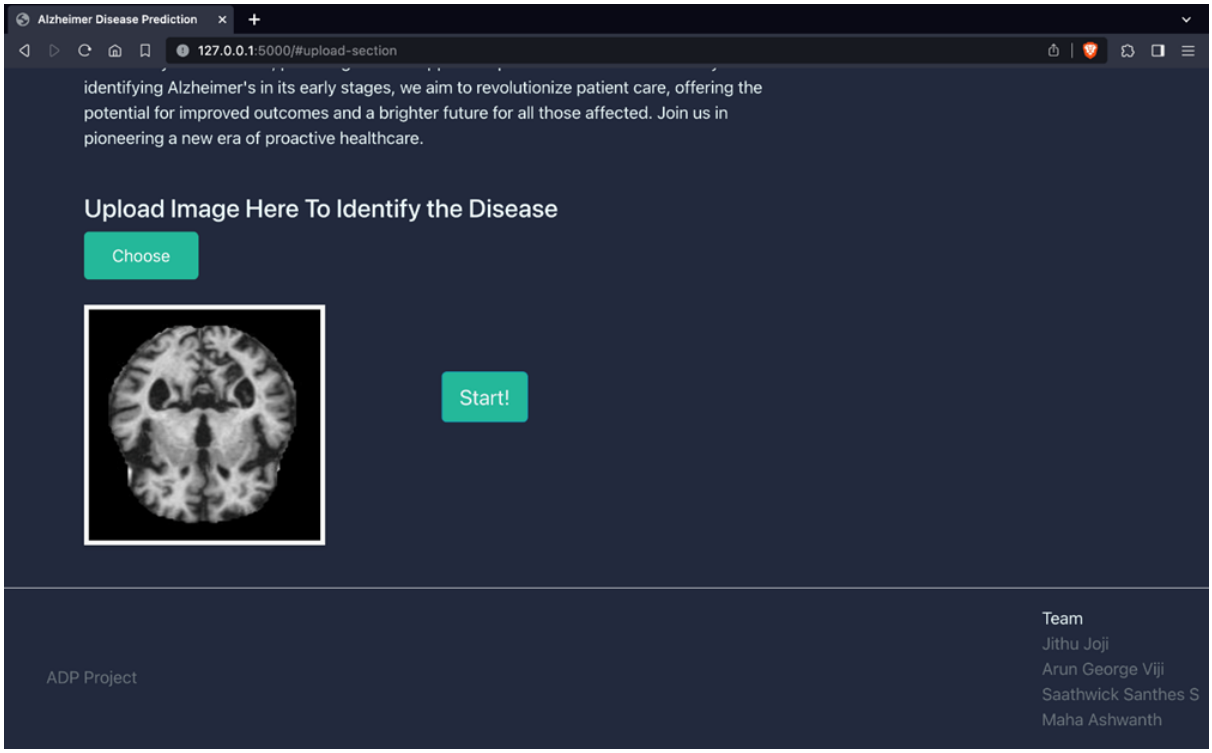
```
 * Serving Flask app 'app1'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

The home page looks like as shown below. When you click on the Predict button, you will be redirected to the predict section.



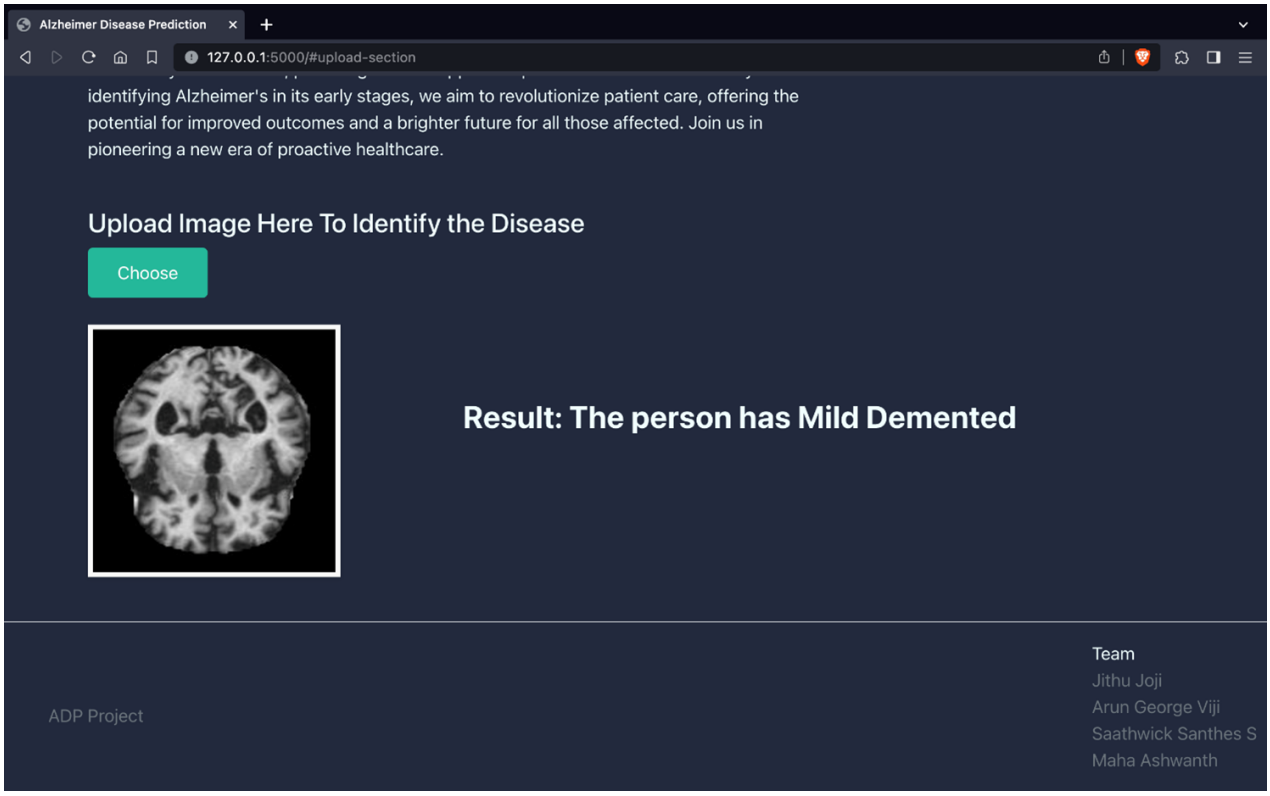

Alzheimer Disease Prediction
127.0.0.1:5000/#upload-section
Revolutionizing Alzheimer's Detection: Early Intervention with Deep Learning Technology
Explore the frontier of medical innovation with our project, leveraging advanced deep learning models. Our cutting-edge approach analyzes medical imaging data, enabling the early detection of subtle signs indicative of Alzheimer's disease, even before symptoms reach a severe stage. This breakthrough technology empowers healthcare providers to offer timely interventions, providing crucial support for patients and their families. By identifying Alzheimer's in its early stages, we aim to revolutionize patient care, offering the potential for improved outcomes and a brighter future for all those affected. Join us in pioneering a new era of proactive healthcare.
Upload Image Here To Identify the Disease
Choose
ADP Project
Team
Jithu Joji
Arun George Viji
Saathwick Santhes S
Maha Ashwanth

Input :

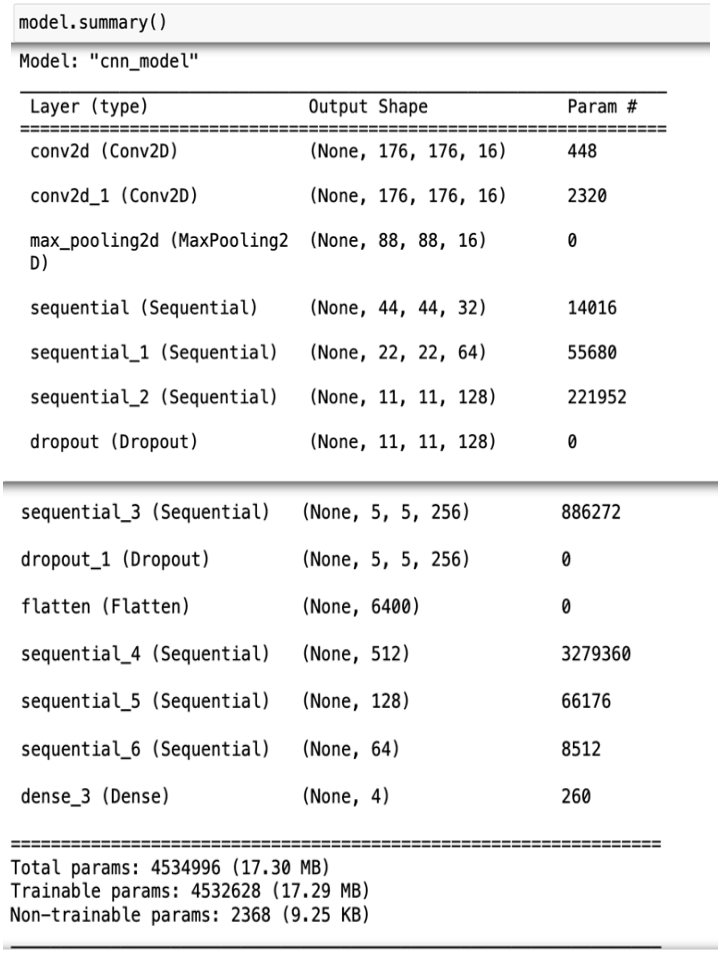
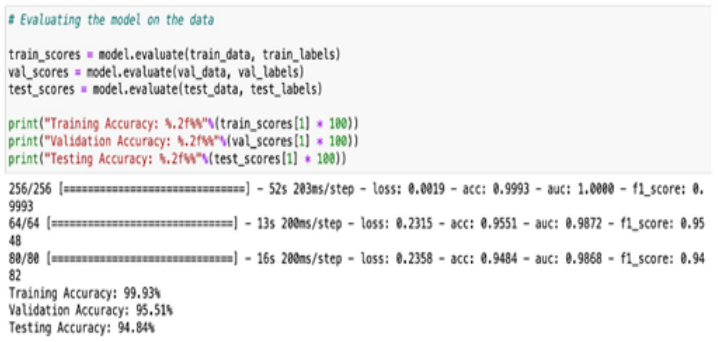

Alzheimer Disease Prediction
127.0.0.1:5000/#upload-section
identifying Alzheimer's in its early stages, we aim to revolutionize patient care, offering the potential for improved outcomes and a brighter future for all those affected. Join us in pioneering a new era of proactive healthcare.
Upload Image Here To Identify the Disease
Choose
Start!
ADP Project
Team
Jithu Joji
Arun George Viji
Saathwick Santhes S
Maha Ashwanth

Once you upload the image and click on the Start button, the output will be displayed as below.

Output :



8. Performance Testing

S.No.	Parameter	Values	Screenshot
1	Model Summary	Total params: 4534996 Trainable params: 4532628 Non-trainable params: 2368	
2	Accuracy	Training Accuracy: 99.93% Validation Accuracy: 95.51% Testing Accuracy: 94.84%	

9. Results

9.1 Output Screenshots

Import model building libraries

```
import numpy as np
import pandas as pd
import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

from tensorflow.keras import Sequential,Input
from tensorflow.keras.layers import Conv2D, Dropout
from tensorflow.keras.layers import BatchNormalization, MaxPool2D
from tensorflow.keras.layers import Dense, Flatten
import tensorflow_addons as tfa

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

Image Preprocessing

```
import os
import shutil

dataset_path = r"Alzheimer_s Dataset"

combined_dataset_path = r"Dataset"

# Create the combined dataset folder if it doesn't exist
os.makedirs(combined_dataset_path, exist_ok=True)

# List of class folders to combine
class_folders = ["VeryMildDemented", "NonDemented", "MildDemented", "ModerateDemented"]

# Iterate through the "train" and "test" folders
for folder_name in ["train", "test"]:
    folder_path = os.path.join(dataset_path, folder_name)

    # Iterate through the class folders
    for class_folder in class_folders:
        class_path = os.path.join(folder_path, class_folder)

        # Iterate through the files in each class folder
        for file_name in os.listdir(class_path):
            file_path = os.path.join(class_path, file_name)

            # Copy the file to the combined dataset folder
            destination_path = os.path.join(combined_dataset_path, class_folder, file_name)
            os.makedirs(os.path.dirname(destination_path), exist_ok=True)
            shutil.copy(file_path, destination_path)
```

```
#2. Configure image data generator
IMG_SIZE = 176
IMAGE_SIZE = [176,176]
DIM = (IMG_SIZE, IMG_SIZE)

ZOOM = [.99, 1.01]
BRIGHTNESS = [0.8, 1.2]
FILL_MODE = "constant"
DATA_FORMAT = "channels_last"
train_datagen = ImageDataGenerator(rescale = 1./255, brightness_range=BRIGHTNESS, zoom_range=ZOOM, data_format=DATA_FORMAT, fill_
```

```
#3. Apply image data generator functionality to train and test images
x_train = train_datagen.flow_from_directory(r"C:\Users\chris\Downloads\ADP Project\Dataset",target_size=DIM, batch_size=6500, shu
```

```
Found 6400 images belonging to 4 classes.
```

```
train_data, train_labels = x_train.next()
print(train_data.shape, train_labels.shape)
```

```
(6400, 176, 176, 3) (6400, 4)
```

```
print(x_train.class_indices)
```

```
{'MildDemented': 0, 'ModerateDemented': 1, 'NonDemented': 2, 'VeryMildDemented': 3}
```

```
#Performing over-sampling of the data, since the classes are imbalanced
sm = SMOTE(random_state=42)
train_data, train_labels = sm.fit_resample(train_data.reshape(-1, IMG_SIZE * IMG_SIZE * 3), train_labels)
train_data = train_data.reshape(-1, IMG_SIZE, IMG_SIZE, 3)
print(train_data.shape, train_labels.shape)
```

```
(12800, 176, 176, 3) (12800, 4)
```

```
#Splitting the data into train, test, and validation sets
train_data, test_data, train_labels, test_labels = train_test_split(train_data, train_labels, test_size = 0.2, random_state=42)
train_data, val_data, train_labels, val_labels = train_test_split(train_data, train_labels, test_size = 0.2, random_state=42)
```

Model Building

```
def conv_block(filters, act='relu'):
    """Defining a Convolutional NN block for a Sequential CNN model. """

    block = Sequential()
    block.add(Conv2D(filters, 3, activation=act, padding='same'))
    block.add(Conv2D(filters, 3, activation=act, padding='same'))
    block.add(BatchNormalization())
    block.add(MaxPool2D())

    return block
```

```
def dense_block(units, dropout_rate, act='relu'):
    """Defining a Dense NN block for a Sequential CNN model. """

    block = Sequential()
    block.add(Dense(units, activation=act))
    block.add(BatchNormalization())
    block.add(Dropout(dropout_rate))

    return block
```

```
def construct_model(act='relu'):
    """Constructing a Sequential CNN architecture for performing the classification task. """

    model = Sequential([
        Input(shape=(*IMAGE_SIZE, 3)),
        Conv2D(16, 3, activation=act, padding='same'),
        Conv2D(16, 3, activation=act, padding='same'),
        MaxPool2D(),
        conv_block(32),
        conv_block(64),
        conv_block(128),
        Dropout(0.2),
        conv_block(256),
        Dropout(0.2),
        Flatten(),
        dense_block(512, 0.7),
        dense_block(128, 0.5),
        dense_block(64, 0.3),
        Dense(4, activation='softmax')
    ], name = "cnn_model")

    return model
```

```
model = construct_model()
```

```
METRICS = [tf.keras.metrics.CategoricalAccuracy(name='acc'),tf.keras.metrics.AUC(name='auc'),tfa.metrics.F1Score(num_classes=4)]
model.compile(optimizer='adam',loss=tf.losses.CategoricalCrossentropy(),metrics=METRICS)
```

```
#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('val_acc') > 0.99:
            print("\nReached accuracy threshold! Terminating training.")
            self.model.stop_training = True

my_callback = MyCallback()
```

```
#Fit the training data to the model and validate it using the validation data
model.fit(train_data, train_labels, validation_data=(val_data, val_labels),callbacks=my_callback,epochs=100)
```

```
Epoch 1/100
256/256 [==============================] - 145s 562ms/step - loss: 1.6891 - acc: 0.2765 - auc: 0.5282 - f1_score:
0.2753 - val_loss: 1.9973 - val_acc: 0.2422 - val_auc: 0.5185 - val_f1_score: 0.0975
Epoch 2/100
256/256 [==============================] - 145s 567ms/step - loss: 1.4286 - acc: 0.3192 - auc: 0.5847 - f1_score:
0.3145 - val_loss: 1.5013 - val_acc: 0.2393 - val_auc: 0.5012 - val_f1_score: 0.0987
Epoch 3/100
256/256 [==============================] - 816s 3s/step - loss: 0.9941 - acc: 0.5436 - auc: 0.8183 - f1_score: 0.52
02 - val_loss: 1.8700 - val_acc: 0.4146 - val_auc: 0.6506 - val_f1_score: 0.3658
Epoch 4/100
256/256 [==============================] - 153s 595ms/step - loss: 0.7979 - acc: 0.6385 - auc: 0.8830 - f1_score:
0.6302 - val_loss: 2.1478 - val_acc: 0.4775 - val_auc: 0.7073 - val_f1_score: 0.3594
Epoch 5/100
256/256 [==============================] - 155s 605ms/step - loss: 0.6883 - acc: 0.6904 - auc: 0.9125 - f1_score:
0.6841 - val_loss: 1.1839 - val_acc: 0.4956 - val_auc: 0.8323 - val_f1_score: 0.3769
Epoch 6/100
256/256 [==============================] - 233s 912ms/step - loss: 0.6539 - acc: 0.7031 - auc: 0.9213 - f1_score:
0.6997 - val_loss: 1.0413 - val_acc: 0.6060 - val_auc: 0.8701 - val_f1_score: 0.5719
```

```
Epoch 7/100
256/256 [==============================] - 216s 845ms/step - loss: 0.6184 - acc: 0.7233 - auc: 0.9297 - f1_score:
0.7227 - val_loss: 0.6159 - val_acc: 0.6958 - val_auc: 0.9264 - val_f1_score: 0.6964
Epoch 8/100
256/256 [==============================] - 226s 884ms/step - loss: 0.5485 - acc: 0.7570 - auc: 0.9445 - f1_score:
0.7564 - val_loss: 0.6865 - val_acc: 0.6328 - val_auc: 0.9048 - val_f1_score: 0.6020
Epoch 9/100
256/256 [==============================] - 251s 981ms/step - loss: 0.5481 - acc: 0.7615 - auc: 0.9447 - f1_score:
0.7612 - val_loss: 0.8633 - val_acc: 0.6333 - val_auc: 0.8720 - val_f1_score: 0.6429
Epoch 10/100
256/256 [==============================] - 223s 871ms/step - loss: 0.4852 - acc: 0.7883 - auc: 0.9562 - f1_score:
0.7888 - val_loss: 0.6715 - val_acc: 0.6777 - val_auc: 0.9150 - val_f1_score: 0.6657
Epoch 11/100
256/256 [==============================] - 213s 834ms/step - loss: 0.4496 - acc: 0.8059 - auc: 0.9627 - f1_score:
0.8060 - val_loss: 0.4677 - val_acc: 0.7920 - val_auc: 0.9592 - val_f1_score: 0.7865
Epoch 12/100
256/256 [==============================] - 219s 854ms/step - loss: 0.4074 - acc: 0.8285 - auc: 0.9689 - f1_score:
0.8288 - val_loss: 0.8827 - val_acc: 0.6479 - val_auc: 0.8891 - val_f1_score: 0.5976
```

```
Epoch 13/100
256/256 [==============================] - 218s 850ms/step - loss: 0.4339 - acc: 0.8164 - auc: 0.9654 - f1_score:
0.8166 - val_loss: 0.5375 - val_acc: 0.7305 - val_auc: 0.9532 - val_f1_score: 0.6629
Epoch 14/100
256/256 [==============================] - 219s 855ms/step - loss: 0.3637 - acc: 0.8467 - auc: 0.9752 - f1_score:
0.8462 - val_loss: 1.6647 - val_acc: 0.5508 - val_auc: 0.8326 - val_f1_score: 0.4944
Epoch 15/100
256/256 [==============================] - 213s 832ms/step - loss: 0.3477 - acc: 0.8601 - auc: 0.9774 - f1_score:
0.8601 - val_loss: 1.2978 - val_acc: 0.5728 - val_auc: 0.8516 - val_f1_score: 0.5450
Epoch 16/100
256/256 [==============================] - 214s 836ms/step - loss: 0.3018 - acc: 0.8799 - auc: 0.9827 - f1_score:
0.8797 - val_loss: 0.4002 - val_acc: 0.8330 - val_auc: 0.9729 - val_f1_score: 0.8344
Epoch 17/100
256/256 [==============================] - 233s 910ms/step - loss: 0.2894 - acc: 0.8827 - auc: 0.9841 - f1_score:
0.8825 - val_loss: 0.6789 - val_acc: 0.7505 - val_auc: 0.9537 - val_f1_score: 0.7156
Epoch 18/100
256/256 [==============================] - 246s 962ms/step - loss: 0.2828 - acc: 0.8903 - auc: 0.9848 - f1_score:
0.8901 - val_loss: 0.3758 - val_acc: 0.8521 - val_auc: 0.9766 - val_f1_score: 0.8523
```

```
Epoch 19/100
256/256 [==============================] - 252s 986ms/step - loss: 0.2316 - acc: 0.9147 - auc: 0.9893 - f1_score:
0.9145 - val_loss: 0.5368 - val_acc: 0.7900 - val_auc: 0.9605 - val_f1_score: 0.7639
Epoch 20/100
256/256 [==============================] - 241s 941ms/step - loss: 0.1983 - acc: 0.9264 - auc: 0.9922 - f1_score:
0.9264 - val_loss: 0.8691 - val_acc: 0.7163 - val_auc: 0.9227 - val_f1_score: 0.7108
Epoch 21/100
256/256 [==============================] - 240s 939ms/step - loss: 0.1885 - acc: 0.9327 - auc: 0.9928 - f1_score:
0.9326 - val_loss: 0.6804 - val_acc: 0.7798 - val_auc: 0.9473 - val_f1_score: 0.7735
Epoch 22/100
256/256 [==============================] - 239s 934ms/step - loss: 0.1591 - acc: 0.9409 - auc: 0.9947 - f1_score:
0.9407 - val_loss: 0.2856 - val_acc: 0.8896 - val_auc: 0.9861 - val_f1_score: 0.8877
Epoch 23/100
256/256 [==============================] - 248s 967ms/step - loss: 0.1460 - acc: 0.9482 - auc: 0.9951 - f1_score:
0.9482 - val_loss: 1.2427 - val_acc: 0.6699 - val_auc: 0.8628 - val_f1_score: 0.6741
Epoch 24/100
256/256 [==============================] - 243s 948ms/step - loss: 0.1459 - acc: 0.9476 - auc: 0.9951 - f1_score:
0.9476 - val_loss: 0.4146 - val_acc: 0.8491 - val_auc: 0.9790 - val_f1_score: 0.8425
```

```
Epoch 25/100
256/256 [==============================] - 292s 1s/step - loss: 0.1275 - acc: 0.9585 - auc: 0.9959 - f1_score: 0.95
84 - val_loss: 0.1938 - val_acc: 0.9277 - val_auc: 0.9928 - val_f1_score: 0.9274
Epoch 26/100
256/256 [==============================] - 273s 1s/step - loss: 0.1140 - acc: 0.9590 - auc: 0.9972 - f1_score: 0.95
89 - val_loss: 0.3242 - val_acc: 0.8940 - val_auc: 0.9834 - val_f1_score: 0.8938
Epoch 27/100
256/256 [==============================] - 251s 980ms/step - loss: 0.1071 - acc: 0.9626 - auc: 0.9971 - f1_score:
0.9626 - val_loss: 0.2522 - val_acc: 0.9077 - val_auc: 0.9902 - val_f1_score: 0.9056
Epoch 28/100
256/256 [==============================] - 233s 909ms/step - loss: 0.0856 - acc: 0.9739 - auc: 0.9978 - f1_score:
0.9739 - val_loss: 0.3217 - val_acc: 0.8882 - val_auc: 0.9831 - val_f1_score: 0.8898
Epoch 29/100
256/256 [==============================] - 235s 918ms/step - loss: 0.0934 - acc: 0.9691 - auc: 0.9975 - f1_score:
0.9691 - val_loss: 0.3955 - val_acc: 0.8828 - val_auc: 0.9802 - val_f1_score: 0.8757
Epoch 30/100
256/256 [==============================] - 247s 965ms/step - loss: 0.0907 - acc: 0.9697 - auc: 0.9975 - f1_score:
0.9697 - val_loss: 0.3866 - val_acc: 0.8745 - val_auc: 0.9816 - val_f1_score: 0.8702
```

```
Epoch 31/100
256/256 [==============================] - 237s 926ms/step - loss: 0.0759 - acc: 0.9764 - auc: 0.9983 - f1_score:
0.9764 - val_loss: 0.2729 - val_acc: 0.9116 - val_auc: 0.9872 - val_f1_score: 0.9101
Epoch 32/100
256/256 [==============================] - 211s 825ms/step - loss: 0.0983 - acc: 0.9657 - auc: 0.9976 - f1_score:
0.9657 - val_loss: 0.2741 - val_acc: 0.9043 - val_auc: 0.9880 - val_f1_score: 0.9059
Epoch 33/100
256/256 [==============================] - 219s 855ms/step - loss: 0.0694 - acc: 0.9766 - auc: 0.9988 - f1_score:
0.9766 - val_loss: 0.2249 - val_acc: 0.9268 - val_auc: 0.9907 - val_f1_score: 0.9258
Epoch 34/100
256/256 [==============================] - 213s 833ms/step - loss: 0.0746 - acc: 0.9761 - auc: 0.9979 - f1_score:
0.9761 - val_loss: 0.5122 - val_acc: 0.8633 - val_auc: 0.9691 - val_f1_score: 0.8593
Epoch 35/100
256/256 [==============================] - 213s 832ms/step - loss: 0.0665 - acc: 0.9795 - auc: 0.9982 - f1_score:
0.9795 - val_loss: 0.2069 - val_acc: 0.9331 - val_auc: 0.9917 - val_f1_score: 0.9327
Epoch 36/100
256/256 [==============================] - 212s 826ms/step - loss: 0.0673 - acc: 0.9797 - auc: 0.9982 - f1_score:
0.9797 - val_loss: 0.3220 - val_acc: 0.9004 - val_auc: 0.9843 - val_f1_score: 0.8989
```

```
Epoch 37/100
256/256 [==============================] - 213s 832ms/step - loss: 0.0702 - acc: 0.9773 - auc: 0.9985 - f1_score:
0.9773 - val_loss: 0.2536 - val_acc: 0.9238 - val_auc: 0.9869 - val_f1_score: 0.9242
Epoch 38/100
256/256 [==============================] - 218s 853ms/step - loss: 0.0454 - acc: 0.9861 - auc: 0.9993 - f1_score:
0.9861 - val_loss: 0.2527 - val_acc: 0.9331 - val_auc: 0.9845 - val_f1_score: 0.9324
Epoch 39/100
256/256 [==============================] - 214s 836ms/step - loss: 0.0564 - acc: 0.9816 - auc: 0.9986 - f1_score:
0.9816 - val_loss: 0.3924 - val_acc: 0.8921 - val_auc: 0.9762 - val_f1_score: 0.8933

Epoch 40/100
256/256 [==============================] - 207s 810ms/step - loss: 0.0514 - acc: 0.9829 - auc: 0.9990 - f1_score:
0.9829 - val_loss: 1.1440 - val_acc: 0.7554 - val_auc: 0.9148 - val_f1_score: 0.7481
Epoch 41/100
256/256 [==============================] - 203s 792ms/step - loss: 0.0538 - acc: 0.9832 - auc: 0.9990 - f1_score:
0.9832 - val_loss: 0.2511 - val_acc: 0.9253 - val_auc: 0.9899 - val_f1_score: 0.9243
Epoch 42/100
256/256 [==============================] - 215s 840ms/step - loss: 0.0508 - acc: 0.9839 - auc: 0.9991 - f1_score:
0.9839 - val_loss: 0.5120 - val_acc: 0.8608 - val_auc: 0.9717 - val_f1_score: 0.8501
```

```
Epoch 43/100
256/256 [==============================] - 201s 784ms/step - loss: 0.0578 - acc: 0.9822 - auc: 0.9986 - f1_score:
0.9822 - val_loss: 0.2956 - val_acc: 0.9048 - val_auc: 0.9865 - val_f1_score: 0.9033
Epoch 44/100
256/256 [==============================] - 200s 781ms/step - loss: 0.0605 - acc: 0.9799 - auc: 0.9983 - f1_score:
0.9798 - val_loss: 0.1963 - val_acc: 0.9390 - val_auc: 0.9920 - val_f1_score: 0.9388
Epoch 45/100
256/256 [==============================] - 203s 793ms/step - loss: 0.0465 - acc: 0.9854 - auc: 0.9992 - f1_score:
0.9853 - val_loss: 0.1754 - val_acc: 0.9512 - val_auc: 0.9921 - val_f1_score: 0.9507
Epoch 46/100
256/256 [==============================] - 204s 797ms/step - loss: 0.0302 - acc: 0.9910 - auc: 0.9996 - f1_score:
0.9910 - val_loss: 0.6197 - val_acc: 0.8555 - val_auc: 0.9593 - val_f1_score: 0.8549
Epoch 47/100
256/256 [==============================] - 197s 769ms/step - loss: 0.0794 - acc: 0.9725 - auc: 0.9979 - f1_score:
0.9725 - val_loss: 0.2373 - val_acc: 0.9297 - val_auc: 0.9894 - val_f1_score: 0.9285
Epoch 48/100
256/256 [==============================] - 195s 761ms/step - loss: 0.0374 - acc: 0.9879 - auc: 0.9995 - f1_score:
0.9879 - val_loss: 0.2649 - val_acc: 0.9292 - val_auc: 0.9846 - val_f1_score: 0.9293
```

```
Epoch 49/100
256/256 [==============================] - 196s 766ms/step - loss: 0.0338 - acc: 0.9890 - auc: 0.9993 - f1_score:
0.9890 - val_loss: 0.1791 - val_acc: 0.9473 - val_auc: 0.9923 - val_f1_score: 0.9468
Epoch 50/100
256/256 [==============================] - 209s 815ms/step - loss: 0.0491 - acc: 0.9847 - auc: 0.9989 - f1_score:
0.9847 - val_loss: 0.1678 - val_acc: 0.9521 - val_auc: 0.9918 - val_f1_score: 0.9522
Epoch 51/100
256/256 [==============================] - 179s 700ms/step - loss: 0.0329 - acc: 0.9907 - auc: 0.9995 - f1_score:
0.9907 - val_loss: 0.2764 - val_acc: 0.9258 - val_auc: 0.9859 - val_f1_score: 0.9241
Epoch 52/100
256/256 [==============================] - 179s 700ms/step - loss: 0.0335 - acc: 0.9894 - auc: 0.9995 - f1_score:
0.9894 - val_loss: 0.2164 - val_acc: 0.9414 - val_auc: 0.9895 - val_f1_score: 0.9406
Epoch 53/100
256/256 [==============================] - 187s 729ms/step - loss: 0.0410 - acc: 0.9869 - auc: 0.9993 - f1_score:
0.9869 - val_loss: 0.5101 - val_acc: 0.8667 - val_auc: 0.9631 - val_f1_score: 0.8676
Epoch 54/100
256/256 [==============================] - 193s 755ms/step - loss: 0.0475 - acc: 0.9856 - auc: 0.9988 - f1_score:
0.9856 - val_loss: 0.3262 - val_acc: 0.9165 - val_auc: 0.9787 - val_f1_score: 0.9156


Epoch 55/100
256/256 [==============================] - 195s 761ms/step - loss: 0.0360 - acc: 0.9882 - auc: 0.9995 - f1_score:
0.9882 - val_loss: 0.2790 - val_acc: 0.9263 - val_auc: 0.9854 - val_f1_score: 0.9256
Epoch 56/100
256/256 [==============================] - 191s 747ms/step - loss: 0.0228 - acc: 0.9938 - auc: 0.9997 - f1_score:
0.9938 - val_loss: 0.3429 - val_acc: 0.9233 - val_auc: 0.9771 - val_f1_score: 0.9233
Epoch 57/100
256/256 [==============================] - 190s 742ms/step - loss: 0.0303 - acc: 0.9899 - auc: 0.9994 - f1_score:
0.9899 - val_loss: 0.2657 - val_acc: 0.9307 - val_auc: 0.9845 - val_f1_score: 0.9297
Epoch 58/100
256/256 [==============================] - 182s 710ms/step - loss: 0.0430 - acc: 0.9862 - auc: 0.9990 - f1_score:
0.9862 - val_loss: 0.1925 - val_acc: 0.9458 - val_auc: 0.9917 - val_f1_score: 0.9459
Epoch 59/100
256/256 [==============================] - 183s 715ms/step - loss: 0.0351 - acc: 0.9888 - auc: 0.9994 - f1_score:
0.9888 - val_loss: 0.8623 - val_acc: 0.7896 - val_auc: 0.9312 - val_f1_score: 0.7827
Epoch 60/100
256/256 [==============================] - 183s 715ms/step - loss: 0.0373 - acc: 0.9880 - auc: 0.9994 - f1_score:
0.9880 - val_loss: 0.2484 - val_acc: 0.9331 - val_auc: 0.9877 - val_f1_score: 0.9322


Epoch 61/100
256/256 [==============================] - 184s 719ms/step - loss: 0.0360 - acc: 0.9901 - auc: 0.9991 - f1_score:
0.9901 - val_loss: 0.1849 - val_acc: 0.9507 - val_auc: 0.9916 - val_f1_score: 0.9501
Epoch 62/100
256/256 [==============================] - 197s 770ms/step - loss: 0.0193 - acc: 0.9941 - auc: 0.9997 - f1_score:
0.9941 - val_loss: 0.4036 - val_acc: 0.9077 - val_auc: 0.9757 - val_f1_score: 0.9070
Epoch 63/100
256/256 [==============================] - 201s 785ms/step - loss: 0.0340 - acc: 0.9901 - auc: 0.9992 - f1_score:
0.9901 - val_loss: 0.2027 - val_acc: 0.9448 - val_auc: 0.9896 - val_f1_score: 0.9450
Epoch 64/100
256/256 [==============================] - 200s 783ms/step - loss: 0.0439 - acc: 0.9861 - auc: 0.9990 - f1_score:
0.9861 - val_loss: 0.1760 - val_acc: 0.9521 - val_auc: 0.9913 - val_f1_score: 0.9522
Epoch 65/100
256/256 [==============================] - 201s 784ms/step - loss: 0.0369 - acc: 0.9897 - auc: 0.9994 - f1_score:
0.9897 - val_loss: 0.1864 - val_acc: 0.9473 - val_auc: 0.9909 - val_f1_score: 0.9472
Epoch 66/100
256/256 [==============================] - 203s 792ms/step - loss: 0.0375 - acc: 0.9877 - auc: 0.9990 - f1_score:
0.9877 - val_loss: 0.1647 - val_acc: 0.9541 - val_auc: 0.9925 - val_f1_score: 0.9539


Epoch 67/100
256/256 [==============================] - 202s 789ms/step - loss: 0.0201 - acc: 0.9940 - auc: 0.9997 - f1_score:
0.9940 - val_loss: 0.2174 - val_acc: 0.9453 - val_auc: 0.9893 - val_f1_score: 0.9452
Epoch 68/100
256/256 [==============================] - 204s 797ms/step - loss: 0.0295 - acc: 0.9911 - auc: 0.9994 - f1_score:
0.9911 - val_loss: 0.1949 - val_acc: 0.9492 - val_auc: 0.9909 - val_f1_score: 0.9489
Epoch 69/100
256/256 [==============================] - 204s 797ms/step - loss: 0.0307 - acc: 0.9908 - auc: 0.9994 - f1_score:
0.9908 - val_loss: 0.1685 - val_acc: 0.9507 - val_auc: 0.9924 - val_f1_score: 0.9504
Epoch 70/100
256/256 [==============================] - 204s 798ms/step - loss: 0.0330 - acc: 0.9891 - auc: 0.9991 - f1_score:
0.9891 - val_loss: 0.1688 - val_acc: 0.9536 - val_auc: 0.9922 - val_f1_score: 0.9538
Epoch 71/100
256/256 [==============================] - 205s 802ms/step - loss: 0.0260 - acc: 0.9927 - auc: 0.9995 - f1_score:
0.9927 - val_loss: 0.1847 - val_acc: 0.9507 - val_auc: 0.9917 - val_f1_score: 0.9502
Epoch 72/100
256/256 [==============================] - 204s 799ms/step - loss: 0.0182 - acc: 0.9946 - auc: 0.9996 - f1_score:
0.9946 - val_loss: 0.2271 - val_acc: 0.9443 - val_auc: 0.9869 - val_f1_score: 0.9447
```

```
Epoch 73/100
256/256 [==============================] - 205s 800ms/step - loss: 0.0256 - acc: 0.9932 - auc: 0.9993 - f1_score:
0.9932 - val_loss: 0.1973 - val_acc: 0.9463 - val_auc: 0.9900 - val_f1_score: 0.9459
Epoch 74/100
256/256 [==============================] - 206s 803ms/step - loss: 0.0250 - acc: 0.9919 - auc: 0.9996 - f1_score:
0.9919 - val_loss: 0.1898 - val_acc: 0.9497 - val_auc: 0.9896 - val_f1_score: 0.9499
Epoch 75/100
256/256 [==============================] - 206s 805ms/step - loss: 0.0200 - acc: 0.9930 - auc: 0.9998 - f1_score:
0.9930 - val_loss: 0.2569 - val_acc: 0.9419 - val_auc: 0.9854 - val_f1_score: 0.9422
Epoch 76/100
256/256 [==============================] - 207s 808ms/step - loss: 0.0175 - acc: 0.9950 - auc: 0.9996 - f1_score:
0.9950 - val_loss: 0.3128 - val_acc: 0.9287 - val_auc: 0.9804 - val_f1_score: 0.9287
Epoch 77/100
256/256 [==============================] - 207s 810ms/step - loss: 0.0396 - acc: 0.9886 - auc: 0.9987 - f1_score:
0.9886 - val_loss: 0.1571 - val_acc: 0.9541 - val_auc: 0.9944 - val_f1_score: 0.9538
Epoch 78/100
256/256 [==============================] - 208s 813ms/step - loss: 0.0163 - acc: 0.9957 - auc: 0.9997 - f1_score:
0.9957 - val_loss: 0.2138 - val_acc: 0.9482 - val_auc: 0.9885 - val_f1_score: 0.9479


Epoch 79/100
256/256 [==============================] - 206s 806ms/step - loss: 0.0170 - acc: 0.9946 - auc: 0.9997 - f1_score:
0.9946 - val_loss: 0.2035 - val_acc: 0.9507 - val_auc: 0.9880 - val_f1_score: 0.9506
Epoch 80/100
256/256 [==============================] - 205s 802ms/step - loss: 0.0230 - acc: 0.9937 - auc: 0.9996 - f1_score:
0.9937 - val_loss: 0.2214 - val_acc: 0.9414 - val_auc: 0.9883 - val_f1_score: 0.9405
Epoch 81/100
256/256 [==============================] - 208s 811ms/step - loss: 0.0277 - acc: 0.9922 - auc: 0.9994 - f1_score:
0.9922 - val_loss: 0.2643 - val_acc: 0.9287 - val_auc: 0.9857 - val_f1_score: 0.9292
Epoch 82/100
256/256 [==============================] - 205s 802ms/step - loss: 0.0252 - acc: 0.9918 - auc: 0.9994 - f1_score:
0.9918 - val_loss: 0.2049 - val_acc: 0.9507 - val_auc: 0.9875 - val_f1_score: 0.9509
Epoch 83/100
256/256 [==============================] - 205s 800ms/step - loss: 0.0206 - acc: 0.9938 - auc: 0.9998 - f1_score:
0.9938 - val_loss: 0.2573 - val_acc: 0.9370 - val_auc: 0.9828 - val_f1_score: 0.9374
Epoch 84/100
256/256 [==============================] - 205s 801ms/step - loss: 0.0151 - acc: 0.9956 - auc: 0.9998 - f1_score:
0.9956 - val_loss: 0.1761 - val_acc: 0.9526 - val_auc: 0.9912 - val_f1_score: 0.9525


Epoch 85/100
256/256 [==============================] - 206s 804ms/step - loss: 0.0232 - acc: 0.9933 - auc: 0.9995 - f1_score:
0.9933 - val_loss: 0.1840 - val_acc: 0.9502 - val_auc: 0.9918 - val_f1_score: 0.9501
Epoch 86/100
256/256 [==============================] - 211s 826ms/step - loss: 0.0170 - acc: 0.9944 - auc: 0.9997 - f1_score:
0.9944 - val_loss: 0.2619 - val_acc: 0.9443 - val_auc: 0.9863 - val_f1_score: 0.9441
Epoch 87/100
256/256 [==============================] - 207s 808ms/step - loss: 0.0157 - acc: 0.9958 - auc: 0.9996 - f1_score:
0.9958 - val_loss: 0.2902 - val_acc: 0.9365 - val_auc: 0.9817 - val_f1_score: 0.9361
Epoch 88/100
256/256 [==============================] - 206s 806ms/step - loss: 0.0236 - acc: 0.9927 - auc: 0.9996 - f1_score:
0.9927 - val_loss: 0.1584 - val_acc: 0.9590 - val_auc: 0.9924 - val_f1_score: 0.9589
Epoch 89/100
256/256 [==============================] - 205s 801ms/step - loss: 0.0222 - acc: 0.9932 - auc: 0.9996 - f1_score:
0.9932 - val_loss: 1.0883 - val_acc: 0.7988 - val_auc: 0.9231 - val_f1_score: 0.7939
Epoch 90/100
256/256 [==============================] - 204s 798ms/step - loss: 0.0312 - acc: 0.9904 - auc: 0.9994 - f1_score:
0.9904 - val_loss: 0.1729 - val_acc: 0.9561 - val_auc: 0.9897 - val_f1_score: 0.9562


Epoch 91/100
256/256 [==============================] - 205s 801ms/step - loss: 0.0188 - acc: 0.9943 - auc: 0.9996 - f1_score:
0.9943 - val_loss: 0.1855 - val_acc: 0.9541 - val_auc: 0.9916 - val_f1_score: 0.9540
Epoch 92/100
256/256 [==============================] - 209s 816ms/step - loss: 0.0124 - acc: 0.9963 - auc: 0.9999 - f1_score:
0.9963 - val_loss: 0.2140 - val_acc: 0.9521 - val_auc: 0.9888 - val_f1_score: 0.9523
Epoch 93/100
256/256 [==============================] - 206s 807ms/step - loss: 0.0123 - acc: 0.9973 - auc: 0.9997 - f1_score:
0.9973 - val_loss: 0.3036 - val_acc: 0.9375 - val_auc: 0.9874 - val_f1_score: 0.9367
Epoch 94/100
256/256 [==============================] - 208s 812ms/step - loss: 0.0185 - acc: 0.9944 - auc: 0.9996 - f1_score:
0.9944 - val_loss: 0.3299 - val_acc: 0.9375 - val_auc: 0.9854 - val_f1_score: 0.9366
Epoch 95/100
256/256 [==============================] - 210s 820ms/step - loss: 0.0392 - acc: 0.9884 - auc: 0.9988 - f1_score:
0.9884 - val_loss: 0.1867 - val_acc: 0.9497 - val_auc: 0.9911 - val_f1_score: 0.9492
Epoch 96/100
256/256 [==============================] - 215s 839ms/step - loss: 0.0120 - acc: 0.9965 - auc: 0.9999 - f1_score:
0.9965 - val_loss: 0.2038 - val_acc: 0.9521 - val_auc: 0.9881 - val_f1_score: 0.9522
```

```
Epoch 97/100
256/256 [==============================] - 212s 830ms/step - loss: 0.0150 - acc: 0.9957 - auc: 0.9997 - f1_score:
0.9957 - val_loss: 0.2031 - val_acc: 0.9521 - val_auc: 0.9885 - val_f1_score: 0.9518
Epoch 98/100
256/256 [==============================] - 205s 802ms/step - loss: 0.0188 - acc: 0.9945 - auc: 0.9996 - f1_score:
0.9945 - val_loss: 0.1918 - val_acc: 0.9546 - val_auc: 0.9887 - val_f1_score: 0.9546
Epoch 99/100
256/256 [==============================] - 204s 795ms/step - loss: 0.0188 - acc: 0.9943 - auc: 0.9998 - f1_score:
0.9943 - val_loss: 0.2093 - val_acc: 0.9521 - val_auc: 0.9873 - val_f1_score: 0.9516
Epoch 100/100
256/256 [==============================] - 204s 796ms/step - loss: 0.0089 - acc: 0.9972 - auc: 1.0000 - f1_score:
0.9972 - val_loss: 0.2315 - val_acc: 0.9551 - val_auc: 0.9872 - val_f1_score: 0.9548

<keras.src.callbacks.History at 0x28a0286d0>
```

```
model.summary()
```

```
Model: "cnn_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 176, 176, 16)      448

 conv2d_1 (Conv2D)           (None, 176, 176, 16)      2320

 max_pooling2d (MaxPooling2  (None, 88, 88, 16)        0
 D)

 sequential (Sequential)     (None, 44, 44, 32)        14016

 sequential_1 (Sequential)   (None, 22, 22, 64)        55680

 sequential_2 (Sequential)   (None, 11, 11, 128)       221952

 dropout (Dropout)           (None, 11, 11, 128)       0


 sequential_3 (Sequential)   (None, 5, 5, 256)         886272

 dropout_1 (Dropout)         (None, 5, 5, 256)         0

 flatten (Flatten)           (None, 6400)              0

 sequential_4 (Sequential)   (None, 512)               3279360

 sequential_5 (Sequential)   (None, 128)               66176

 sequential_6 (Sequential)   (None, 64)                8512

 dense_3 (Dense)             (None, 4)                 260

=================================================================
Total params: 4534996 (17.30 MB)
Trainable params: 4532628 (17.29 MB)
Non-trainable params: 2368 (9.25 KB)
_________________________________________________________________
```

```
# Evaluating the model on the data

train_scores = model.evaluate(train_data, train_labels)
val_scores = model.evaluate(val_data, val_labels)
test_scores = model.evaluate(test_data, test_labels)

print("Training Accuracy: %.2f%%"%(train_scores[1] * 100))
print("Validation Accuracy: %.2f%%"%(val_scores[1] * 100))
print("Testing Accuracy: %.2f%%"%(test_scores[1] * 100))
```

```
256/256 [==============================] - 52s 203ms/step - loss: 0.0019 - acc: 0.9993 - auc: 1.0000 - f1_score: 0.9993
64/64 [==============================] - 13s 200ms/step - loss: 0.2315 - acc: 0.9551 - auc: 0.9872 - f1_score: 0.9548
80/80 [==============================] - 16s 200ms/step - loss: 0.2358 - acc: 0.9484 - auc: 0.9868 - f1_score: 0.9482
Training Accuracy: 99.93%
Validation Accuracy: 95.51%
Testing Accuracy: 94.84%
```

```
#save model
model.save('adp.h5')
```

```
#Predicting the test data
pred = model.predict(test_data)
```

```
80/80 [==============================] - 16s 196ms/step
```

```
pred_ls = np.argmax(pred, axis=1)
test_ls = np.argmax(test_labels, axis=1)
print(classification_report(test_ls, pred_ls))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.96       639
           1       1.00      1.00      1.00       635
           2       0.94      0.89      0.91       662
           3       0.92      0.91      0.91       624

    accuracy                           0.95      2560
   macro avg       0.95      0.95      0.95      2560
weighted avg       0.95      0.95      0.95      2560
```

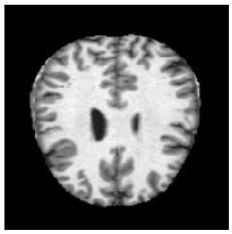
```
confusion_matrix(test_ls, pred_ls)
```

```
array([[636,   0,   0,   3],
       [  0, 635,   0,   0],
       [ 25,   0, 590,  47],
       [ 19,   0,  38, 567]])
```

Testing the Model

```
model = load_model(r"adp.h5",compile=False)
```

```
img = image.load_img(r"Dataset\NonDemented\32 (54).jpg",target_size=(176,176))
img
```



```
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
```

```
pred = model.predict(x)
pred_class = np.argmax(pred,axis=1)
index = ['MildDemented','ModerateDemented','NonDemented','VeryMildDemented']
result = str(index[pred_class[0]])
result
```

```
1/1 [==============================] - 0s 194ms/step
```

```
'NonDemented'
```

10. Advantages and Disadvantages

Advantages

1. **Superior Accuracy**: The model excels in predicting Alzheimer’s disease using neuroimaging data, offering dependable results for early diagnosis and intervention.
2. **Robust Feature Extraction**: The deep learning architecture of the model facilitates the extraction of complex features from neuroimaging data, enabling the detection of subtle patterns linked to Alzheimer’s disease progression.
3. **Scalability**: The system is scalable and can effectively manage a large amount of neuroimaging data, making it ideal for large-scale clinical studies and practical healthcare applications.

Disadvantages:

1. **Data Requirement**: The model necessitates a significant volume of labelled data for training. This could be a constraint when the availability of annotated neuroimaging data is limited.
2. **Computational Complexity**: The training process for the model can be computationally demanding, requiring substantial processing power and time. This could pose a potential challenge for resource-limited environments.
3. **Interpretability**: The inherent complexity of deep learning models like Xception can make it difficult to interpret the model’s decision-making process, potentially hindering the understanding of the underlying biological mechanisms of Alzheimer’s disease.

11. Conclusion

In conclusion, the utilization of the CNN model for predicting Alzheimer's disease presents promising outcomes, demonstrating high accuracy and robust capabilities in feature extraction. Despite challenges related to data requirements and computational complexity, the model's scalability and potential for early diagnosis establish it as a valuable tool in the ongoing endeavors to combat Alzheimer's disease.

12. Future Scope

The future trajectory of this research involves further refining the interpretability of the CNN model, exploring methodologies to diminish data requirements for training, and optimizing computational efficiency to facilitate practical implementation across diverse healthcare settings. Additionally, the integration of multimodal data and the exploration of transfer learning techniques could augment the model's performance, extending its applicability in neurodegenerative disease research and clinical practice. Collaborations with neuroscientists and healthcare professionals remain pivotal, paving the way for the integration of this technology into routine clinical practice for early detection of Alzheimer's disease and personalized treatment planning.

13. Appendix

13.1 Source Code

Program Code

```
#Import model building libraries

import numpy as np
import pandas as pd
import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

from tensorflow.keras import Sequential,Input
from tensorflow.keras.layers import Conv2D, Dropout
from tensorflow.keras.layers import BatchNormalization, MaxPool2D
from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

#Image Preprocessing

import os
import shutil

# Define the source and destination directories
source_dir = r"Alzheimer_s Dataset"
destination_dir = r"Dataset"

# Create the destination directory if it doesn't exist
if not os.path.exists(destination_dir):
    os.makedirs(destination_dir)

# Define the folder names
folders = ["VeryMildDemented", "ModerateDemented", "NonDemented",
"MildDemented"]

# Loop through the folders
for folder in folders:
    # Create the corresponding folder in the destination directory
    destination_folder = os.path.join(destination_dir, folder)
    if not os.path.exists(destination_folder):
        os.makedirs(destination_folder)

    # Loop through the train and test folders
    for subfolder in ["train", "test"]:
        subfolder_path = os.path.join(source_dir, subfolder, folder)

        # Check if the subfolder exists
        if os.path.exists(subfolder_path):
            # Loop through the files in the subfolder
```

```
            for file_name in os.listdir(subfolder_path):
                file_path = os.path.join(subfolder_path, file_name)

                # Copy the file to the corresponding folder in the
destination directory
                destination_file_path = os.path.join(destination_folder,
file_name)
                shutil.copy(file_path, destination_file_path)

#2. Configure image data generator
IMG_SIZE = 176
IMAGE_SIZE = [176, 176]
DIM = (IMG_SIZE, IMG_SIZE)

ZOOM = [.99, 1.01]
BRIGHT_RANGE = [0.8, 1.2]
FILL_MODE = "constant"
DATA_FORMAT = "channels_last"
train_datagen = ImageDataGenerator(rescale = 1./255,
brightness_range=BRIGHT_RANGE, zoom_range=ZOOM, data_format=DATA_FORMAT,
fill_mode=FILL_MODE, horizontal_flip=True)

#3. Apply image data generator functionality to train and test images
x_train =
train_datagen.flow_from_directory(r'/content/Dataset',target_size=DIM,
batch_size=6500, shuffle=False)

train_data, train_labels = x_train.next()
print(train_data.shape, train_labels.shape)

print(x_train.class_indices)

#Performing over-sampling of the data, since the classes are imbalanced
sm = SMOTE(random_state=42)
train_data, train_labels = sm.fit_resample(train_data.reshape(-1, IMG_SIZE
* IMG_SIZE * 3), train_labels)
train_data = train_data.reshape(-1, IMG_SIZE, IMG_SIZE, 3)
print(train_data.shape, train_labels.shape)

#Splitting the data into train, test, and validation sets
from sklearn.model_selection import train_test_split
train_data, test_data, train_labels, test_labels =
train_test_split(train_data, train_labels, test_size = 0.2,
random_state=42)
train_data, val_data, train_labels, val_labels =
train_test_split(train_data, train_labels, test_size = 0.2,
random_state=42)

#Model Building

def conv_block(filters, act='relu'):
    """Defining a Convolutional NN block for a Sequential CNN model. """

    block = Sequential()
    block.add(Conv2D(filters, 3, activation=act, padding='same'))
    block.add(Conv2D(filters, 3, activation=act, padding='same'))
    block.add(BatchNormalization())
```

```
    block.add(MaxPool2D())

    return block

def dense_block(units, dropout_rate, act='relu'):
    """Defining a Dense NN block for a Sequential CNN model. """

    block = Sequential()
    block.add(Dense(units, activation=act))
    block.add(BatchNormalization())
    block.add(Dropout(dropout_rate))

    return block

def construct_model(act='relu'):
    """Constructing a Sequential CNN architecture for performing the
classification task. """

    model = Sequential([
        Input(shape=(*IMAGE_SIZE, 3)),
        Conv2D(16, 3, activation=act, padding='same'),
        Conv2D(16, 3, activation=act, padding='same'),
        MaxPool2D(),
        conv_block(32),
        conv_block(64),
        conv_block(128),
        Dropout(0.2),
        conv_block(256),
        Dropout(0.2),
        Flatten(),
        dense_block(512, 0.7),
        dense_block(128, 0.5),
        dense_block(64, 0.3),
        Dense(4, activation='softmax')
    ], name = "cnn_model")

    return model

model = construct_model()
METRICS =
[tf.keras.metrics.CategoricalAccuracy(name='acc'),tf.keras.metrics.AUC(name
='auc'),tfa.metrics.F1Score(num_classes=4)]
model.compile(optimizer='adam',loss=tf.losses.CategoricalCrossentropy(),met
rics=METRICS)

#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('val_acc') > 0.99:
            print("\nReached accuracy threshold! Terminating training.")
            self.model.stop_training = True

my_callback = MyCallback()

#Fit the training data to the model and validate it using the validation
data
```

```
model.fit(train_data, train_labels, validation_data=(val_data,
val_labels),callbacks=my_callback,epochs=100)

model.summary()

#save model
model.save('adp.h5')

# Evaluating the model on the data
train_scores = model.evaluate(train_data, train_labels)
val_scores = model.evaluate(val_data, val_labels)
test_scores = model.evaluate(test_data, test_labels)

print("Training Accuracy: %.2f%%"%(train_scores[1] * 100))
print("Validation Accuracy: %.2f%%"%(val_scores[1] * 100))
print("Testing Accuracy: %.2f%%"%(test_scores[1] * 100))

#Predicting the test data
pred = model.predict(test_data)
pred_ls = np.argmax(pred, axis=1)
test_ls = np.argmax(test_labels, axis=1)
print(classification_report(test_ls, pred_ls))
confusion_matrix(test_ls, pred_ls)
```

#Testing the Model

```
model = load_model(r"adp.h5",compile=False)
img = image.load_img(r"Dataset\NonDemented\32
(54).jpg",target_size=(176,176))
img
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
pred = model.predict(x)
pred_class = np.argmax(pred,axis=1)
index =
['MildDemented','ModerateDemented','NonDemented','VeryMildDemented']
result = str(index[pred_class[0]])
result
```

Flask Code (app1.py)

```
import numpy as np
import os
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask , request, render_template

app = Flask(__name__)

model = load_model("adp.h5",compile=False)

@app.route('/')
def index():
```

```
    return render_template('index.html')

@app.route('/predict',methods = ['GET','POST'])
def upload():
    if request.method == 'POST':
        f = request.files['image']
        print("current path")
        basepath = os.path.dirname(__file__)
        print("current path", basepath)
        filepath = os.path.join(basepath,'uploads',f.filename)
        print("upload folder is ", filepath)
        f.save(filepath)

        img = image.load_img(filepath,target_size = (176,176))
        x = image.img_to_array(img)
        print(x)
        x = np.expand_dims(x,axis =0)
        print(x)
        y=model.predict(x)
        preds=np.argmax(y, axis=1)
        print("Prediction",preds)
        index = ['Mild Demented','Moderate Demented','Non Demented','Very
Mild Demented']
        text = "The person is " + str(index[preds[0]])
    return text
if __name__ == '__main__':
    app.run(debug = False, threaded = False)
```

Html Code (index.html)

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Alzheimer Disease Prediction</title>
    <link
href="https://cdn.bootcss.com/bootstrap/4.0.0/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.bootcss.com/popper.js/1.12.9/umd/popper.min.js"></script>
    <script
src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>
    <script
src="https://cdn.bootcss.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>
```

```
    <link href="{{ url_for('static', filename='css/main.css') }}"
rel="stylesheet">
    <style>
        .navbar-brand {
            display: inline-block;
            padding-top: 0.3125rem;
            padding-bottom: 0.3125rem;
            margin-right: 1rem;
            font-size: 2rem;
            line-height: inherit;
            white-space: nowrap;
        }

        .btn-info {
            background-color: #25b89a;
            color: #F1F6F9;
        }

        .btn-info:hover {
            background-color: #156a59;
        }

        .bg-dark {
            background-color: #4B92BB!important;
        }

        #result {
            font-weight: bold;
            color: F1F6F9;
        }

        body {
            background-color: #212A3E;
            color: azure;
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI',
'Roboto', 'Oxygen',
                'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans',
'Helvetica Neue',
                sans-serif;
            -webkit-font-smoothing: antialiased;
            -moz-osx-font-smoothing: grayscale;
        }

            .team-list li {
            padding-right: 10px;
            }

            .pb-3, .py-3 {
```

```
            padding-bottom: 0rem!important;
            }

        #result {
            font-weight: bold;
            color: F1F6F9;
        }
    </style>
</head>

<body>

    <nav class="navbar navbar-dark" style="background-color: #394867;">
        <div class="container">
            <a class="navbar-brand" href="#">Alzheimer Disease Prediction
using CNN</a>
                  <a class="btn btn-info" href="#upload-
section">Predict</a>
        </div>
    </nav>

    <div class="container">
        <div id="content" style="margin-top:2em">
            <div class="row">
                <div class="col-sm-8 bd">
                    <h3>Alzheimer's disease </h3>
                    <br>
                    <p>Alzheimer's disease (AD) is a progressive and
irreversible neurological disorder that affects
                        the brain, leading to memory loss, cognitive
impairment, and changes in behavior and
                        personality. It is the most common cause of
dementia among older adults and is characterized by
                        the buildup of abnormal protein deposits in the
brain, including amyloid plaques and tau
                        tangles.</p>
                    <br>
                    <h3>Revolutionizing Alzheimer's Detection: Early
Intervention with Deep Learning Technology</h3>
                    <br>
                    <p>Explore the frontier of medical innovation with our
project, leveraging advanced deep learning
                        models. Our cutting-edge approach analyzes medical
imaging data, enabling the early detection
                        of subtle signs indicative of Alzheimer's disease,
even before symptoms reach a severe stage.
                        This breakthrough technology empowers healthcare
providers to offer timely interventions,
```

```
                        providing crucial support for patients and their
families. By identifying Alzheimer's in its
                        early stages, we aim to revolutionize patient care,
offering the potential for improved outcomes
                        and a brighter future for all those affected. Join
us in pioneering a new era of proactive
                        healthcare.</p>
                </div>
                <div class="col-sm-4 bd d-flex justify-content-center
align-items-center">
                    <img

src="https://a.storyblok.com/f/176343/740x416/b97226ff46/alzheimers-
disease-the-value-of-accurate-diagnosis-teaser.jpg"
                        height="45%" width="110%">
                </div>

                <div id="upload-section" class="col-sm-12">
                            <br>
                            <h4>Upload Image Here To Identify the
Disease</h4>
                            <div class="row">
                                  <div class="col-md-4">
                                        <form
action="http://localhost:5000/" id="upload-file" method="post"
enctype="multipart/form-data">
                                              <label for="imageUpload"
class="upload-label">
                                                    Choose
                                              </label>
                                              <input type="file"
name="image" id="imageUpload" accept=".png, .jpg, .jpeg">
                                        </form>

                                        <div class="image-section"
style="display:none;">
                                              <div class="img-preview">
                                                    <div
id="imagePreview"></div>
                                              </div>

                                        </div>

                                  </div>
                                  <div class="col-md-8 d-flex align-
items-center">
                                        <div>
                                              <button type="button"
```

```
class="btn btn-info btn-lg" id="btn-predict" style="display:
none;">Start!</button>
                                            </div>
                                            <div class="loader"
style="display:none;"></div>
                                            <h3 class="text-center">
                                                  <span id="result"> </span>
                                            </h3>
                                        </div>
                                    </div>
                                </div>
            </div>
        </div>
    </div>
</body>

<footer class="d-flex flex-wrap justify-content-between align-items-center
py-3 mt-4 border-top">
    <div class="col-md-4 d-flex align-items-center">
      <a href="/" class="mb-3 me-2 mb-md-0 text-muted text-decoration-none
lh-1">
        <svg class="bi" width="30" height="24"><use
xlink:href="#bootstrap"></use></svg>
      </a>
      <span class="mb-3 mb-md-0 text-muted">ADP Project</span>
    </div>

    <ul class="list-unstyled team-list">
            <li>Team</li>
            <li class="text-muted"><a
href="https://www.linkedin.com/in/jithu-joji/" target="_blank">Jithu
Joji</a></li>
            <li class="text-muted"><a
href="https://www.linkedin.com/in/arun-george-viji-34188a183/"
target="_blank">Arun George Viji</a></li>
            <li class="text-muted"><a
href="https://www.linkedin.com/in/saathwick-santhes-a44949299/"
target="_blank">Saathwick Santhes S</a></li>
            <li class="text-muted"><a
href="https://www.linkedin.com/in/mahaashwanth/" target="_blank">Maha
Ashwanth</a></li>
      </ul>



  </footer>

<footer>
```

```
    <script src="{{ url_for('static', filename='js/main.js') }}"
type="text/javascript"></script>
</footer>

</html>
```

CSS Code (main.css)

```
.img-preview {
    width: 256px;
    height: 256px;
    position: relative;
    border: 5px solid #F8F8F8;
    box-shadow: 0px 2px 4px 0px rgba(0, 0, 0, 0.1);
    margin-top: 1em;
    margin-bottom: 1em;
}


.img-preview>div {
    width: 100%;
    height: 100%;
    background-size: 256px 256px;
    background-repeat: no-repeat;
    background-position: center;
}

input[type="file"] {
    display: none;
}

.upload-label{
    display: inline-block;
    padding: 12px 30px;
    background: #25b89a;
    color: #fff;
    font-size: 1em;
    transition: all .4s;
    cursor: pointer;
    border-radius: 0.3rem;
}

.upload-label:hover{
    background: #34495E;
    color: #39D2B4;
    border-radius: 0.3rem;
}
```

```
.loader {
    border: 8px solid #f3f3f3; /* Light grey */
    border-top: 8px solid #3498db; /* Blue */
    border-radius: 50%;
    width: 50px;
    height: 50px;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

JavaScript Code (main.js)

```
$(document).ready(function () {
    // Init
    $('.image-section').hide();
    $('.loader').hide();
    $('#result').hide();
    $('.btn-info').click(function () {
        $('html, body').animate({
            scrollTop: $("#upload-section").offset().top
        }, 1000);
    });

    // Upload Preview
    function readURL(input) {
        if (input.files && input.files[0]) {
            var reader = new FileReader();
            reader.onload = function (e) {
                $('#imagePreview').css('background-image', 'url(' +
e.target.result + ')');
                $('#imagePreview').hide();
                $('#imagePreview').fadeIn(650);
            }
            reader.readAsDataURL(input.files[0]);
        }
    }
    $("#imageUpload").change(function () {
        $('.image-section').show();
        $('#btn-predict').show();
        $('#result').text('');
        $('#result').hide();
        readURL(this);
    });
```

```
    // Predict
    $('#btn-predict').click(function () {
        var form_data = new FormData($('#upload-file')[0]);

        // Show loading animation
        $(this).hide();
        $('.loader').show();

        // Make prediction by calling api /predict
        $.ajax({
            type: 'POST',
            url: '/predict',
            data: form_data,
            contentType: false,
            cache: false,
            processData: false,
            async: true,
            success: function (data) {
                // Get and display the result
                $('.loader').hide();
                $('#result').fadeIn(600);
                $('#result').text(' Result:  ' + data);
                console.log('Success!');
            },
        });
    });

});
```

13.2 Github Link

https://github.com/smartinternz02/SI-GuidedProject-611663-1698727369.git

13.3. Demo link

https://drive.google.com/file/d/1zNreSnTkLuvE9VIj0SD25UB-S4_Ammtd/view?usp=sharing