TUNISIAN REPUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Sciences and Technology

Professional Personal Project

Third Year Engineering Class

Specialty: Software Engineering

CareAI: A Healthcare System Integrating Azure-Powered Machine Learning, Deep Learning Models for Disease Detection, and a Conversational AI Chatbot

---

Prepared by

Ilef Chebil

Taher Nouira

Mrs Taktak Hajer Supervisor at INSAT

School year: 2023 / 2024

Acknowledgement

We would like to take this opportunity to extend our heartfelt gratitude to everyone who has supported and motivated us throughout our end of year project. This achievement would not have been possible without your assistance.

We are deeply grateful to Mrs. Hajer Taktak, our supervisor from the National Institute of Applied Science and Technology. The completion of this investigation and report would not have been achievable without Mrs. Taktak's guidance, collaboration, and valuable insights.

We also want to express our appreciation for our shared passion for innovation, which has driven our enthusiasm and inspired us to explore advanced technology. The teamwork and dedication of our project group have been crucial in overcoming challenges and achieving success.

Lastly, we want to thank everyone who has contributed, whether through enlightening discussions, helpful advice, or encouraging words. Your contributions have significantly enhanced our project.

Thank you.

Table of Contents

Figures List

Tables List

Abstract

In recent years, the evolution of artificial intelligence (AI) across various domains has underscored the critical need to revolutionize healthcare. AI's potential to enhance disease prediction, diagnosis, and decision-making for medical professionals is immense. This report details the development and evaluation of an advanced healthcare system designed to detect and classify various medical conditions, including Parkinson's disease, Alzheimer's stages, skin cancer, and brain tumor types. The system leverages machine learning models, such as K-Nearest Neighbors (KNN), and deep learning models employing transfer learning techniques, specifically EfficientNetB0, EfficientNetB3, and VGG19. Datasets from Kaggle were utilized for training and validating the models. The system's performance was rigorously evaluated using key metrics including accuracy, precision, recall, and F1 score. These metrics demonstrate the system's efficacy in accurately identifying and classifying medical diseases, thereby supporting healthcare professionals in their critical tasks. Furthermore, the models were developed and deployed following MLOps principles and integrated into a user-friendly web interface. Our findings indicate that the system not only effectively classifies diseases but also provides valuable insights, significantly contributing to the medical field.

Project Context

Introduction

In the healthcare domain, precise and accurate methodologies are required for detecting and classifying various diseases. In this first chapter, we set the stage for our study by focusing on both the difficulty and the necessity of a system facilitating medical decisions and diagnoses. We then examine current approaches and solutions and delve into the fundamental ideas behind deep learning, machine learning, web integration, and deployment. Next, we outline our suggested methodology for developing such a system.

1.1 Problem Statement

The detection and classification of medical diseases require skilled medical staff with ample experience and a discerning eye in the immensely complicated discipline of healthcare. Although a conventional method, human observation comes with its own set of benefits and drawbacks. On the positive side, individuals can recognize patterns and anomalies that strict algorithms might overlook. Over time, experienced doctors develop a sharp intuition and expertise that can surpass machine interpretation. However, the drawback of human observation is its requirement for a significant amount of work, time, and its susceptibility to mistakes. The quality of the image and the observer’s skill all impact how MRI images and patients' conditions should be interpreted. Even merely detecting diseases, let alone distinguishing between different types and stages, poses a formidable challenge even for trained professionals.

1.2 Existing Solutions

Many techniques have been developed over time to reduce the burden of manual diagnosis.

Existing solutions in the healthcare AI landscape, such as IBM Watson Health, have made significant strides in revolutionizing medical decision-making and patient care. IBM Watson Health leverages artificial intelligence and advanced analytics to analyze vast amounts of medical data. Other notable healthcare AI solutions include Google Health, which focuses on leveraging AI for medical imaging analysis and personalized patient care. Additionally, startups like Babylon Health offer virtual healthcare services, allowing patients to consult with healthcare professionals remotely through AI-powered chatbots. [1]

While these AI healthcare solutions offer numerous benefits, including improved efficiency, accuracy, and accessibility, they also come with their own set of drawbacks. Challenges such as data privacy concerns, algorithm biases, and the need for continuous validation and regulatory compliance remain significant challenges. Moreover, the integration of AI technologies into existing healthcare workflows and systems can be complex and time-consuming, requiring careful planning and implementation.

1.3 Proposed Solution

CareAI is a multifaceted healthcare web application built leveraging machine learning and deep learning technologies to address critical medical challenges. The application encompasses multiple components aimed at detecting and predicting various diseases, as well as providing a healthcare chatbot for medical staff interaction.

The main components include skin cancer prediction, brain cancer classification, Alzheimer’s Stages Prediction, Parkinson disease prediction and a healthcare chatbot all made accessible via cloud deployment and a user friendly web application.

Our solution's primary contribution lies in its specialization: each model is tailored to predict a specific disease, thereby mitigating the risk of inaccuracies and errors and delivering results with greater confidence.

1.4 Definition of Concepts

To fully understand the implementation and technological choices made in CareAI, it's essential to define some key concepts and methodologies used throughout the project :

1.4.1 Exploratory Data Analysis

Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods.The main purpose of EDA is to help look at data before making any assumptions. It can help identify obvious errors, as well as better understand patterns within the data, detect outliers or anomalous events, and find interesting relations among the variables, Visualization tools like Matplotlib and Seaborn are employed to create detailed plots that reveal data distributions and relationships. [2]

1.4.2 Machine Learning

Machine Learning is a scientific field, and more specifically a sub-category of artificial intelligence. It consists in letting algorithms discover "patterns", i.e. recurring patterns, in data sets. These data can be numbers, words, images, statistics. Anything that can be stored digitally can be used as data for Machine Learning. By detecting patterns in this data, algorithms learn and improve their performance in the execution of a specific task.In short, Machine Learning algorithms autonomously learn how to perform a task or make predictions from data and improve their performance over time. Once trained, the algorithm will be able to find patterns in new data.[3]

1.4.3 Deep Learning

Deep learning is a specialized subset of machine learning that utilizes artificial neural networks with multiple layers, these are often referred to as deep neural networks. Deep learning models are designed to automatically and adaptively learn to represent data by training on large amounts of data, with little to no manual feature extraction required. Unlike traditional machine learning models that reach a saturation point, deep learning models continue to improve their performance as the size of the data increases, given their ability to learn complex patterns and representations from data.[4] [5]

1.4.4 Classification and Prediction Algorithms

Machine Learning employs a diverse range of classification and prediction algorithms tailored to various challenges. These algorithms can be broadly categorized into supervised and unsupervised learning methods. In supervised learning, models such as decision trees, support vector machines, neural networks, and ensemble methods like random forests and gradient boosting are used to classify data or make predictions based on labeled training data. On the other hand, unsupervised learning algorithms, such as k-means clustering, hierarchical clustering are utilized to uncover hidden patterns or intrinsic structures in unlabeled data.The choice of algorithm depends on the specific problem, data characteristics, and performance requirements, making machine learning a versatile tool across various domains, from healthcare and finance to image recognition and natural language processing.

1.4.5 Natural Language Processing

Natural Language Processing (NLP) plays a vital role in interpreting and analyzing textual data . Utilizing libraries like NLTK, SpaCy, and Transformers (Hugging Face), sophisticated NLP models are trained to extract meaningful insights from texts. These models enable advanced functionalities such as Healthcare Chatbots, which leverage conversational AI platforms like Llama 2 and Langchain to provide accurate and contextually relevant responses to user queries.[6]

1.4.6 Web Integration

1.4.6.1 Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja, and has become one of the most popular Python web application frameworks.[7]

1.4.6.2 React

React is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta and a

community of individual developers and companies.React code is made of entities called components.These components are modular and reusable.React applications typically consist of many layers of components.[8]

1.4.7 Deployment : Azure Cloud Provider

Azure, Microsoft's cloud computing service, offers an extensive array of cloud services, including computing, analytics, storage, and networking. Users can select from these services to develop and scale new applications or run existing ones in the public cloud. Renowned for its unparalleled compute capabilities, Azure is widely regarded as the premier choice for hosting managed machine learning models.[9]

1.5 Dataset Description

- ❖ Parkinson Disease Prediction:

This dataset is a Kaggle dataset that comprises biomedical voice measurements from 31 individuals, 23 of whom have Parkinson's disease (PD). Each row represents one of 195 voice recordings, with columns indicating various voice measures. The primary objective is to differentiate healthy individuals (status = 0) from those with PD (status = 1). Key attributes include measures of fundamental frequency, amplitude variation, noise-to-tonal ratio, nonlinear dynamical complexity, and fundamental frequency variation.

Attribute Information:

Name: ASCII subject name and recording number

MDVP (Hz): Average vocal fundamental frequency

MDVP (Hz): Maximum vocal fundamental frequency

MDVP (Hz): Minimum vocal fundamental frequency

MDVP(%), MDVP(Abs), MDVP, MDVP, Jitter: Variation in fundamental frequency

MDVP, MDVP(dB), Shimmer, MDVP: Variation in amplitude

NHR, HNR: Ratio of noise to tonal components in the voice

status: Health status (1 - PD, 0 - healthy)

RPDE, D2: Nonlinear dynamical complexity measures

DFA: Signal fractal scaling exponent

spread1, spread2, PPE: Nonlinear measures of fundamental frequency variation [10]

❖ Alzheimer's Stages Classification Kaggle Dataset :

Context: The dataset is hand-collected from various verified websites. It is curated to include labels that have been meticulously verified to ensure accuracy.

Content: The dataset consists of MRI images categorized into four classes. Both the training and testing sets include these classes:

1. **Mild Demented**
2. **Moderate Demented**
3. **Non Demented**
4. **Very Mild Demented**

Each class is represented in the training and testing sets, with a total of approximately 5000 images in each set.

Inspiration: The primary goal of sharing this dataset is to facilitate the development of highly accurate models that can predict the stage of Alzheimer's disease.

Each directory contains images segregated according to the severity of Alzheimer's as per the classes mentioned above. [11]

❖ Skin Cancer: Malignant vs Benign Kaggle Dataset:

Context: This dataset is created to help in the development of machine learning models for the classification of skin moles as either benign or malignant.

Content: The dataset consists of two folders:

- Each folder contains 1800 images (224x224 pixels).
- One folder contains images of benign skin moles.
- The other folder contains images of malignant skin moles.

Inspiration: The dataset is shared with the hope that machine learning models trained on it might achieve better and cheaper predictions than those made by dermatologists. The primary aim is to advance the accuracy and accessibility of skin cancer diagnostics.

Usability: The dataset is balanced, making it suitable for training models without requiring additional balancing techniques. The high-quality images and clear classification labels support the development of robust machine learning algorithms for skin cancer detection.[12]

❖ Brain Tumor Types Classification Kaggle Dataset:

Context: This dataset contains MRI images of the brain categorized into normal and three different types of brain tumors. It is designed to facilitate research and development in medical imaging and the application of machine learning techniques for brain tumor classification.

Content: The dataset is organized into four directories, each representing a different class of brain MRI images:

- **Glioma Tumor:** 6307 files
- **Meningioma Tumor:** 6391 files
- **Normal:** 3066 files
- **Pituitary Tumor:** 5908 files

Usability:

The usability score for this dataset is 2.50, indicating potential challenges in terms of data quality or ease of use.

Users may need to perform additional preprocessing to prepare the data for machine learning tasks.

Inspiration:

The primary goal is to provide a comprehensive dataset for training and evaluating machine learning models for the detection and classification of brain tumors. By sharing this dataset, it is hoped that more accurate diagnostic tools can be developed, improving the prognosis and treatment of patients with brain tumors. [13]

1.6 Development Methodology

We adopted a two-fold development methodology for this project. Initially, we embraced an incremental approach, where our primary focus was on developing the models. Subsequently, we aimed to create a responsive and user-friendly web application and integrate the models locally into it. Our secondary vision involved deploying each model seamlessly in the cloud, making them accessible via endpoints using principles of MLOps. Additionally, we adhered to an iterative approach to select the best-performing model for each disease based on predefined metrics. After iterating through multiple models for each sub-disease and identifying the most suitable one, we moved on to the next disease, following the same iterative process. Throughout the development process, we followed the CRISP-DM methodology, which encompasses Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, Deployment, and Monitoring phases for each model. This structured approach ensured thoroughness and efficiency in our development efforts.

Conclusion

In conclusion, the first chapter lays the foundation for our study by emphasizing the necessity of precise methodologies in healthcare for disease detection and classification. We have explored existing solutions, highlighting their advancements and challenges. Our proposed solution, CareAI, offers specialized models tailored to predict specific diseases, aiming to reduce inaccuracies and errors. The development methodology, incorporating incremental and iterative approaches, ensures a thorough and efficient process guided by the CRISP-DM methodology.

Planning Phases

Introduction

The planning phase of the CareAI project is essential for creating a robust and efficient healthcare system. It starts with identifying key users and stakeholders to tailor the system to their needs. This is followed by a detailed functional requirements analysis to outline necessary capabilities like data analysis, model training, deployment, and web integration. Non-functional requirements ensure the system's reliability, security, and efficiency. The use case diagram visualizes user interactions, the class diagram defines system components, and sequence diagrams illustrate interaction flows. Together, these elements provide a clear blueprint for developing a user-friendly and technically sound system.

2.1 Requirement Specification

2.1.1 Actor Identification

In the context of CareAI, healthcare professionals are the primary users of the system. Their interactions with CareAI encompass various aspects of healthcare data analysis, model training, deployment, and utilization. Identifying the specific roles and responsibilities of these professionals helps ensure that the system meets their needs effectively.

2.1 Requirement Analysis

2.2.1 Functional Requirements

CareAI's functional requirements focus on data analysis, model training, deployment, and integration with a web application to facilitate seamless user interactions. The system must support data analysis and preprocessing using Python libraries such as NumPy and Pandas to clean, transform, and normalize healthcare data, making it ready for model training. Model training capabilities should encompass various machine learning algorithms for applications like

disease detection and drug discovery, ensuring these models are scalable and reliable through deployment on Azure Machine Learning Service. The system must provide mechanisms for versioning and managing deployed models to maintain traceability and reproducibility. Integration with a web application is crucial, enabling users to interact with models via a user-friendly interface. Backend components should communicate efficiently with deployed models' API endpoints for data processing and prediction, while frontend components must support user input, data visualization, and the display of model predictions. Additionally, the system should include robust model evaluation and performance monitoring mechanisms to track metrics such as accuracy, latency, and resource utilization, with automated alerts for any anomalies or performance degradation.

2.2.2 Non-Functional Requirements

CareAI's non-functional requirements emphasize performance, reliability, security, and usability to ensure a robust and efficient system. Performance-wise, the system should exhibit low latency and high throughput for model predictions, enabling a seamless user experience even during peak usage. Scalability is essential, with endpoints designed to handle concurrent requests efficiently. In terms of reliability, CareAI must be highly available with minimal downtime for maintenance or updates, ensuring continuous access to the system. Deployed models should be monitored for performance and errors, with automated alerts set up for prompt issue resolution. Security measures are critical, requiring the implementation of stringent protocols to protect sensitive data and prevent unauthorized access. Endpoints exposing the deployed models must enforce authentication and authorization mechanisms. Usability is also a priority; the web application should have an intuitive user interface that is easy to navigate, facilitating user interaction with the deployed models. Comprehensive documentation and user guides are necessary to assist stakeholders and end-users in effectively utilizing the system, ensuring that the solution is accessible and straightforward to operate.

2.1 General Use Case Diagram

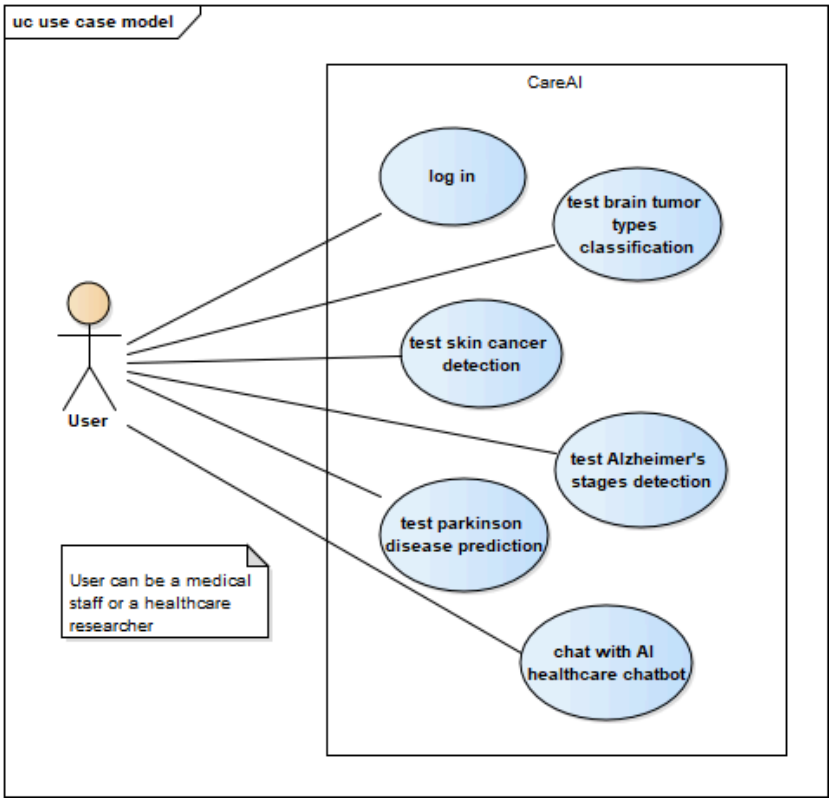

Figure 1 : CareAI class diagram: first release

This use case diagram represents interactions between a user and a healthcare AI system called CareAI. Here are some observations and comments on the diagram:

Use Cases:

- The use cases are clearly defined within the CareAI system:
  - **Log in**: This use case represents the user's ability to access the system.
  - **Test brain tumor types classification**: This use case involves the system classifying different types of brain tumors.
  - **Test skin cancer detection**: This use case involves detecting skin cancer using the system.
  - **Test Alzheimer's stages detection**: This use case involves detecting the stages of Alzheimer's disease.
  - **Test Parkinson disease prediction**: This use case involves predicting Parkinson's disease.
  - **Chat with AI healthcare chatbot**: This use case involves interacting with an AI chatbot for healthcare-related queries.

2.2 Class Diagram

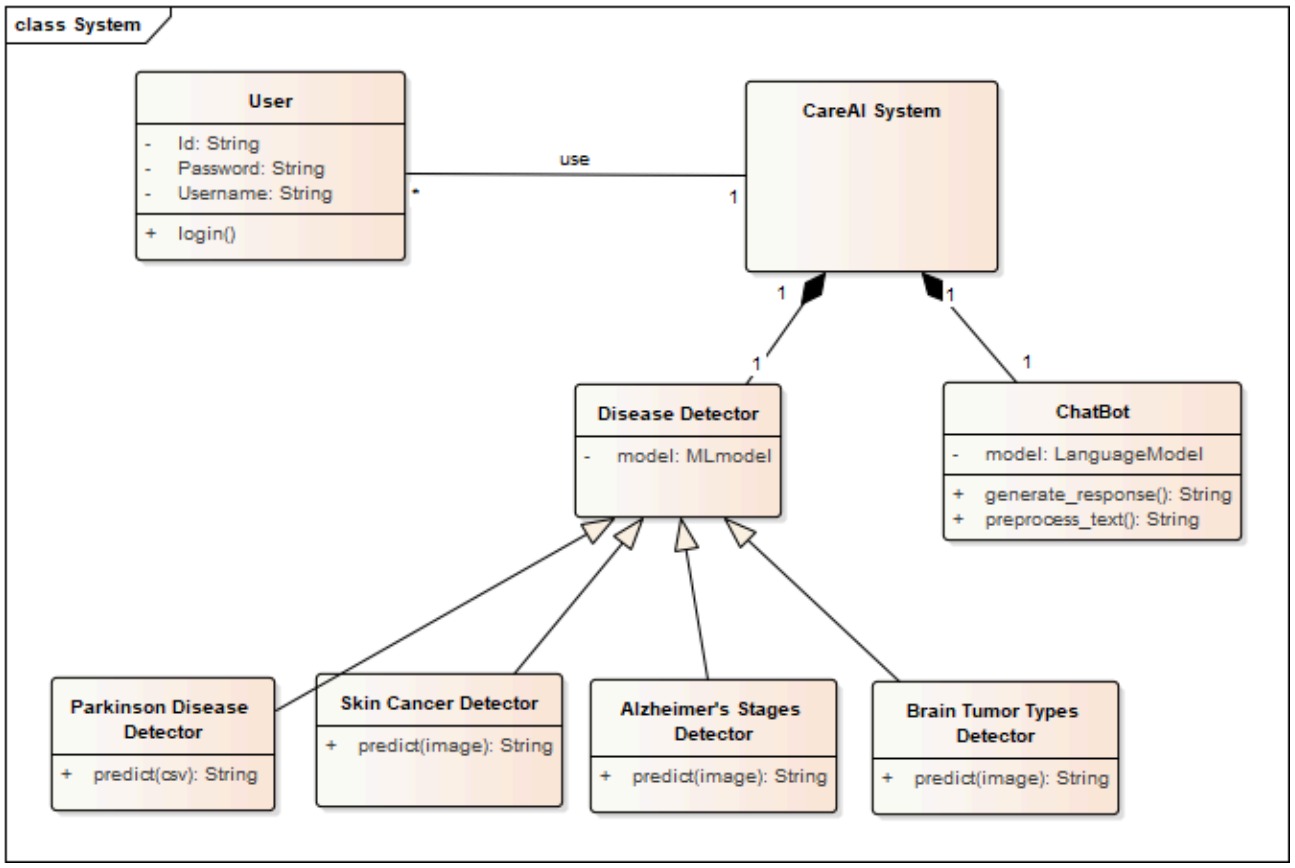

Figure 2: CareAI Class Diagram : First Release

This class diagram provides a detailed view of the CareAI system, including its components and their interactions.

2.3 Sequence Diagram

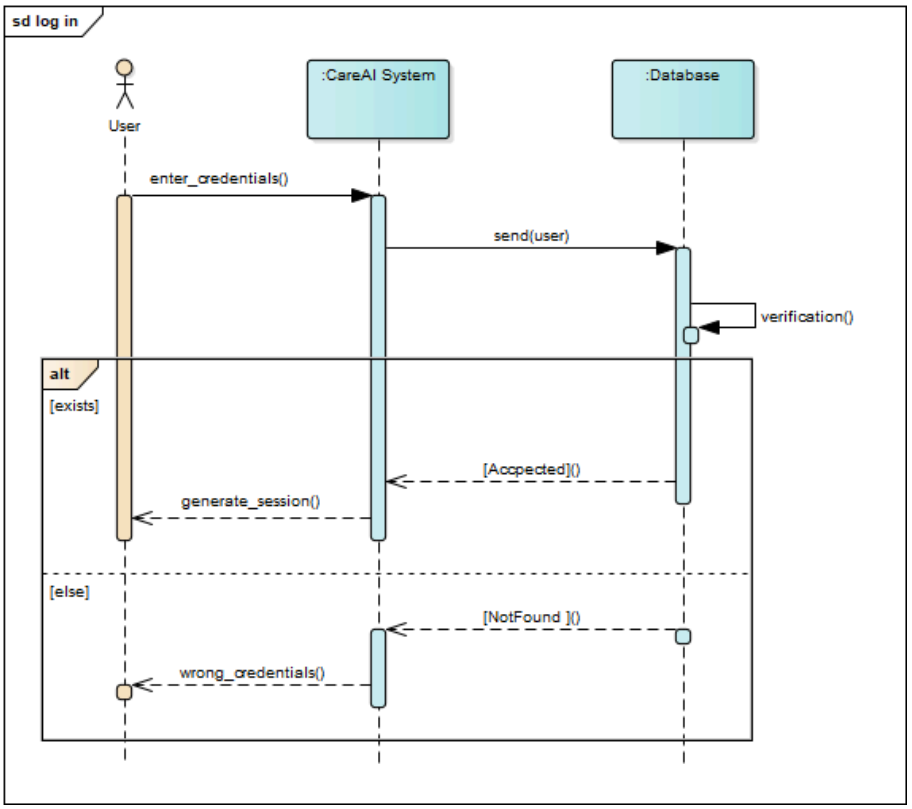

Figure 3 : Login Use Sequence Diagram

This UML use case diagram represents various interactions a User can have with the "CareAI" system. Here's a detailed analysis and commentary on the diagram:

Participants

1. User: The primary actor interacting with the CareAI system. The user can be either a medical staff member or a healthcare researcher.

Use Cases within the CareAI System

1. log in: The user logs into the CareAI system.
2. test brain tumor types classification: The user tests the classification of brain tumor types.
3. test skin cancer detection: The user tests the detection of skin cancer.
4. test Alzheimer's stages detection: The user tests the detection of Alzheimer's disease stages.
5. test Parkinson's disease prediction: The user tests the prediction of Parkinson's disease.

6. chat with AI healthcare chatbot: The user interacts with an AI healthcare chatbot.

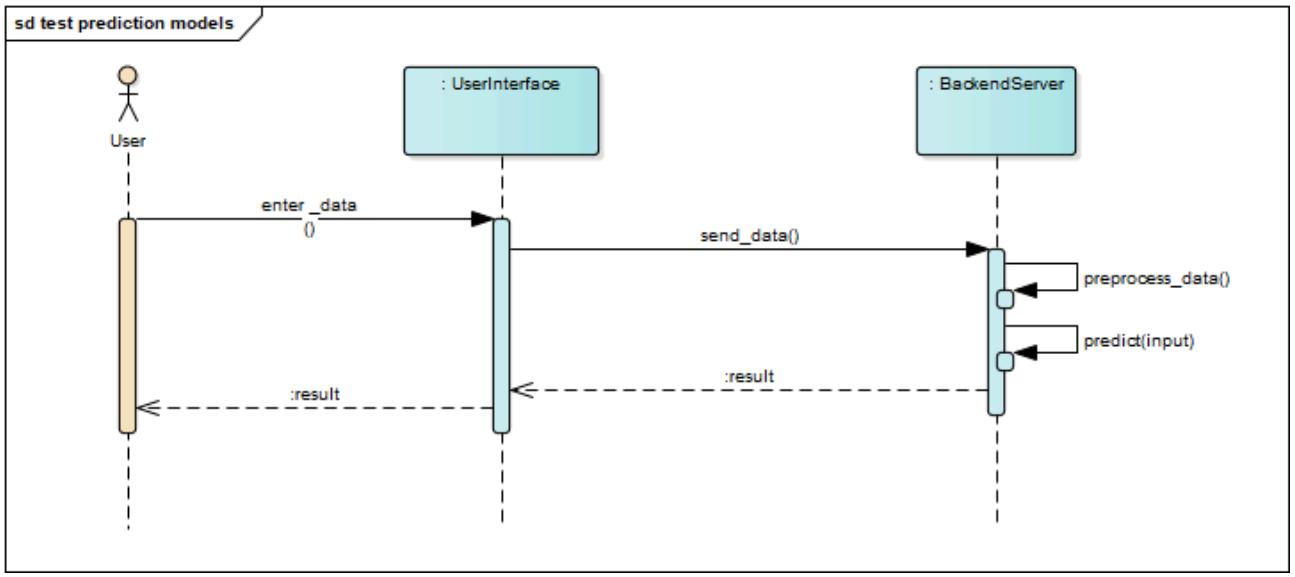

Figure 4 : Test prediction models sequential diagram

This UML sequence diagram represents the interaction between a User, a UserInterface, and a BackendServer for testing prediction models. Here's a detailed analysis and commentary on the diagram:

Participants

1. User: The individual interacting with the system.
2. UserInterface: The front-end interface through which the User enters data.
3. BackendServer: The server responsible for preprocessing data and making predictions.

Interaction Flow

1. enter_data(): The User inputs data into the UserInterface.
2. send_data(): The UserInterface sends the entered data to the BackendServer.
3. preprocess_data(): The BackendServer preprocesses the received data.
4. predict(input): After preprocessing, the BackendServer runs the prediction model using the processed data.
5. result: The prediction result is sent back from the BackendServer to the UserInterface.
6. result: The UserInterface then displays the result to the User.

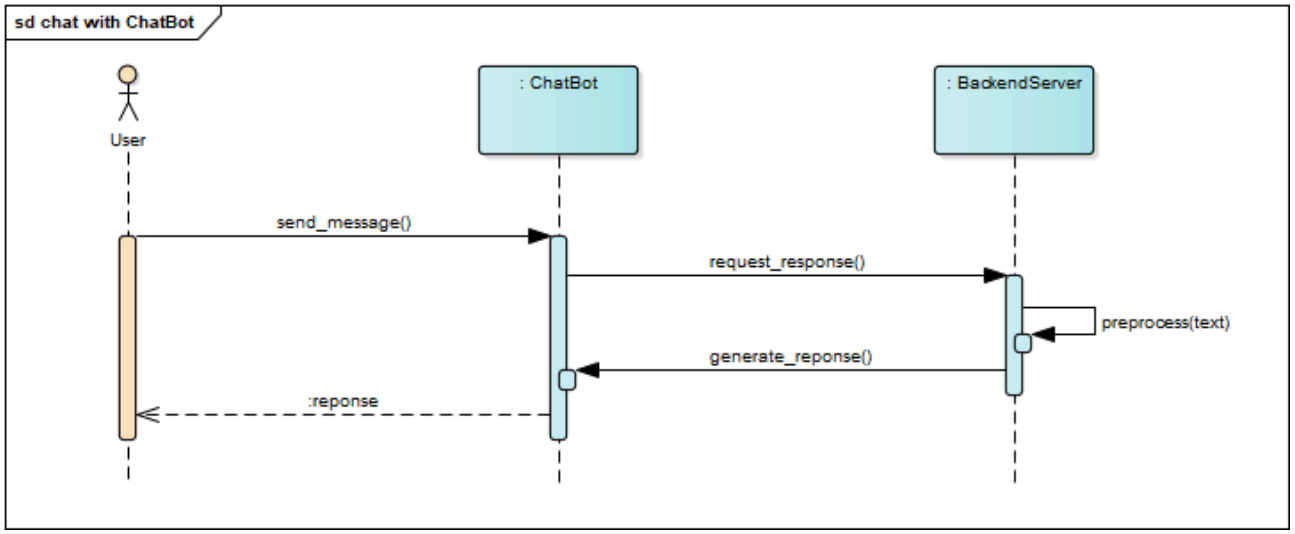

Figure 5 : Converse with AI healthcare chatbot sequential diagram

The diagram you provided is a UML sequence diagram representing the interaction between a User, a ChatBot, and a BackendServer during a chat session. Here's an analysis and commentary on the diagram:

Participants

1. User: The person interacting with the ChatBot.
2. ChatBot: The system receiving and processing messages from the user.
3. BackendServer: The server that assists the ChatBot in generating responses.

Interaction Flow

1. send_message(): The User initiates the interaction by sending a message to the ChatBot.
2. request_response(): The ChatBot receives the message and sends a request to the BackendServer to generate a response.
3. preprocess(text): The BackendServer processes the received text (preprocessing, such as text normalization, tokenization, etc.).
4. generate_response(): The BackendServer generates a response based on the preprocessed text and sends it back to the ChatBot.
5. response: The ChatBot sends the generated response back to the User

2.4 First Release Vision

Our goal for the first release is to develop and deploy four predictive models: Parkinson's disease detection, Alzheimer's stage classification, brain tumor type classification, and skin cancer detection, finally the AI healthcare specialized chatbot. We will use the CRISP-DM methodology to guide the development of these models. Once the models are built, we will compress and optimize them for local deployment.

We will then create a responsive web application using React for the front end and Flask for the back end. The Flask backend will handle the consumption of the predictive models, while the React front end will render the responses. This setup ensures a seamless and efficient user experience, enabling users to access and utilize the predictive models through an intuitive interface.

2.5 Second Release Vision

For the second release, we aim to deploy all models as endpoints in the cloud using Azure ML Studio. This will make the models accessible to all users. We will adhere to MLOps principles to ensure efficient, scalable, and maintainable deployment and operations. These cloud-based endpoints will be integrated into the Flask backend, allowing for seamless consumption of the models within our application. This approach will enhance accessibility, scalability, and reliability of our predictive models.

Conclusion

The planning phase of CareAI establishes a strong foundation for developing an advanced healthcare system. By identifying actors and analyzing functional and non-functional requirements, we ensure the system meets user needs and performs reliably and securely. Use case, class, and sequence diagrams offer clear visualizations of system interactions and structures. This thorough planning is crucial for creating a user-friendly and dependable system, ultimately enhancing healthcare outcomes through innovative technology.

First Release

Introduction

In this third chapter, we delve into the technological choices and implementation strategies behind our healthcare solution. We begin by examining the overall logical architecture, which outlines the high-level design and interactions between various components of the system. Following this, we explore the physical architecture. We also discuss the software environment, including the tools and technologies employed throughout the development process. Subsequently, we cover the implementation phase, providing insights into the practical steps taken to bring the solution to life. We then focus on the design and functionality of the web interface, highlighting how it enhances user experience and accessibility.

3.1 Technological Choices

3.1.1 Overall Logical Architecture

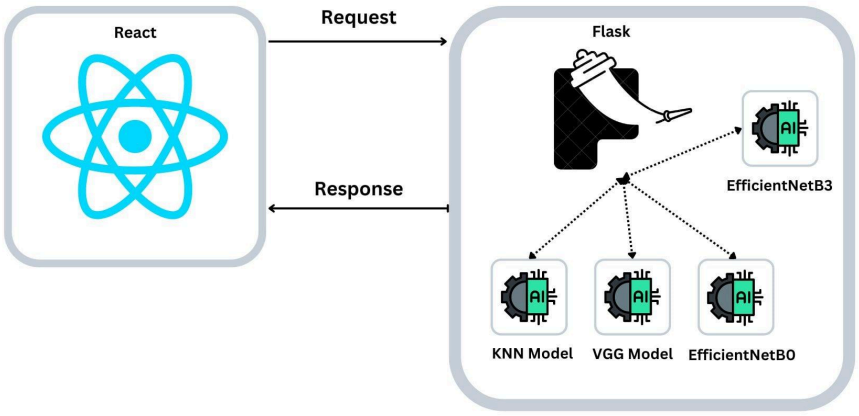

Figure 6 : Flask-React Application Logical Architecture

Components:

1. **User Interface (UI)**

- **React Frontend**: Provides a user-friendly interface for interacting with the system, including input forms for data entry and display of prediction results.

2. **API Layer**
   - **Flask Backend**: Handles API requests from the frontend, processes input data, and communicates with the ML/DL models for predictions.
   - **Machine Learning Models**: Pickled ML models (.pkl files) for traditional machine learning tasks.
   - **Deep Learning Models**: HDF5 DL models (.h5 files) for deep learning tasks.
3. **Data Flow**
   - **Data Input**: Users input data via the React frontend.
   - **Request Handling**: Flask API receives data and processes it.
   - **Model Prediction**: Flask backend loads the appropriate ML/DL models, runs predictions, and returns results.
   - **Data Output**: Prediction results are sent back to the React frontend for display.

Interactions:

1. User Interaction: Users interact with the React frontend through a web browser.
2. API Calls: React frontend sends HTTP requests to the Flask backend.
3. Model Inference: Flask backend processes requests, loads models, performs predictions, and sends responses.
4. Result Display: React frontend displays the prediction results to the user

3.1.2 Physical Architecture

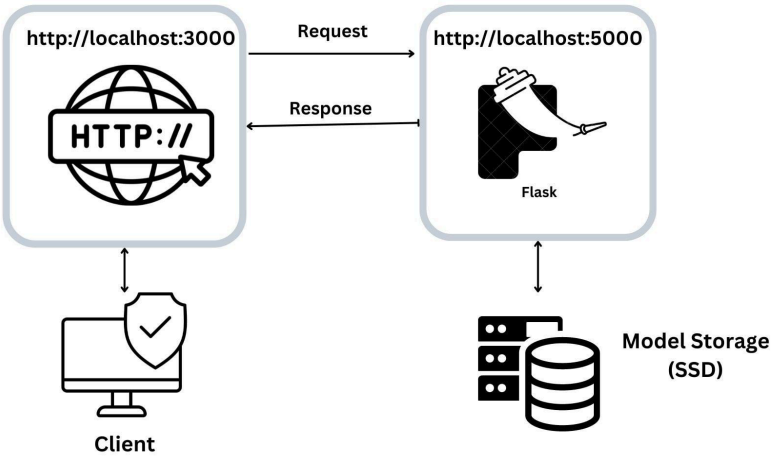

Figure 7 : Flask-React Application Physical Architecture

Components:

1. **Client Machines (Users)**
    - **Devices**: Desktops, laptops, tablets, and smartphones.
    - **Browsers**: Modern web browsers like Chrome, Firefox, Safari, or Edge.
2. **Web Server**
    - React Development Server running on localhost:3000 to serve the React application and act as a reverse proxy to the Flask backend running on localhost:5000..
3. **Storage**
    - **Model Storage**: SSD storage for fast access to ML (.pkl) and DL (.h5) models.

Interactions:

1. **User Requests**: Users send requests to the web server via their browsers.
2. **Reverse Proxy**: The web server routes requests to the appropriate Flask application server.
3. **Model Access**: Flask backend accesses models from local storage for prediction tasks.
4. **Response Delivery**: Prediction results are sent back through the web server to the user's browser.

3.1.3 Software Environment

This software environment ensures a robust, scalable, and maintainable deployment of the React Flask web application, utilizing machine learning and deep learning models for predictions.

- Frontend (React)

- Version: 17.x or 18.x (latest stable version)
- Language: JavaScript/TypeScript
- State Management: Redux
- Routing: React Router

- Styling: SASS, Styled Components, and Material-UI

- Development Tools

- Package Manager: npm (Node Package Manager)
- Build Tool: Create React App

- Backend (Flask)

- Version: 2.x (latest stable version)
- Language: Python 3.8 or later

- Machine Learning and Deep Learning Libraries

- Scikit-Learn: for traditional ML models (.pkl files)
- TensorFlow/Keras: for deep learning models (.h5 files)
- NumPy: for numerical operations
- Pandas: for data manipulation and analysis

- API Development

- Flask-RESTful: for building REST APIs
- Flask-CORS: to handle Cross-Origin Resource Sharing (CORS)

- Data Storage and Serialization

- Pickle: for serializing and deserializing Python object structures (ML models)
- HDF5: for storing large amounts of data (DL models)

- Development Tools

- Virtual Environment: virtualenv or venv for managing Python environments
- Package Manager: pip for installing Python packages

- IDE/Code Editor

- VS Code: Visual Studio Code with relevant extensions for JavaScript and Python

- Version Control

- Git: for version control
- GitHub: for repository hosting and collaboration

3.2 Implementation

Our approach to developing this web application involves a seamless integration of a React frontend with a Flask backend, facilitating a user-friendly interface for medical predictions using both machine learning (ML) and deep learning (DL) models. Initially, we set up the development environment by installing necessary dependencies for both React and Flask. The backend, built with Flask, hosts endpoints that load and serve the ML (.pkl files) and DL (.h5 files) compressed models, responding to prediction requests. Concurrently, the React frontend provides an intuitive interface for users to input data and receive predictions.

This iterative and modular development approach ensures that both components can be independently developed, tested, and integrated, leading to a robust and efficient system for healthcare predictions.

3.3 Web Interface

The web interface is user-friendly and responsive, built using React.js. It enables users to upload CSV files for Parkinson's disease detection and images for Alzheimer's, brain tumor, and skin cancer detection, ensuring ease of use and accessibility.

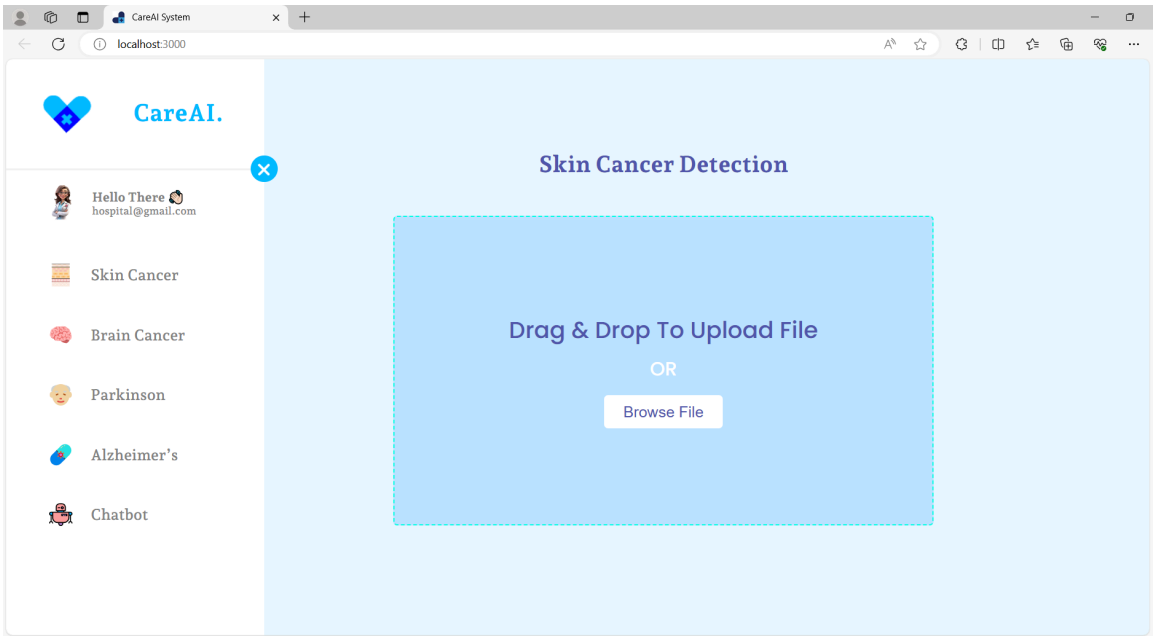

Figure 8 : UI interface for skin cancer detection

Conclusion

In this initial release, we laid the foundation for our healthcare application, making critical technological choices and implementing key components. The logical and physical architectures were carefully designed to ensure efficiency and scalability, while the software environment was meticulously selected to support our development needs. With the implementation phase underway, we've begun integrating ML and DL models into a user-friendly web interface, empowering users to interact seamlessly with our predictive system. This marks a significant step towards our goal of delivering accurate and accessible healthcare solutions.

Second Release

Introduction

In this chapter, we delve into the essential components of our project, beginning with the technological choices that encompass the overall logical and physical architectures, and the software environment. We then explore the CRISP-DM methodology, detailing each phase from business understanding and data preparation to modeling and evaluation. The chapter also covers the specific models developed for Parkinson's disease, Alzheimer's stages, brain tumor stages, and skin cancer detection. Finally, we discuss the deployment process using Azure Machine Learning Studio, including datastore, model registry, environments, and endpoints, concluding with monitoring strategies.

4.1. Technological Choices

In our quest to elevate our healthcare application and make it universally accessible to medical staff, we needed to rethink our technology choices to embark on a transformative migration of our local system to the cloud, with scalability as a paramount objective. We have settled on Azure as our premier cloud provider to host our managed machine learning models, ensuring unparalleled computational power and global accessibility.

4.1.1 Overall Logical Architecture

In this level, data scientists can reproduce their training model workflows by others typically using pipeline technologies. Assets and metadata are captured automatically.

The workflow of training models is defined by a pipeline which captures complex job dependencies (but in our case the training model workflow is simple, so we don't have to use it).

Job manages your training job, logs parameters, and metrics.

Finally, an environment defines Python packages, environment variables, and Docker settings for training model and scoring model.[14]

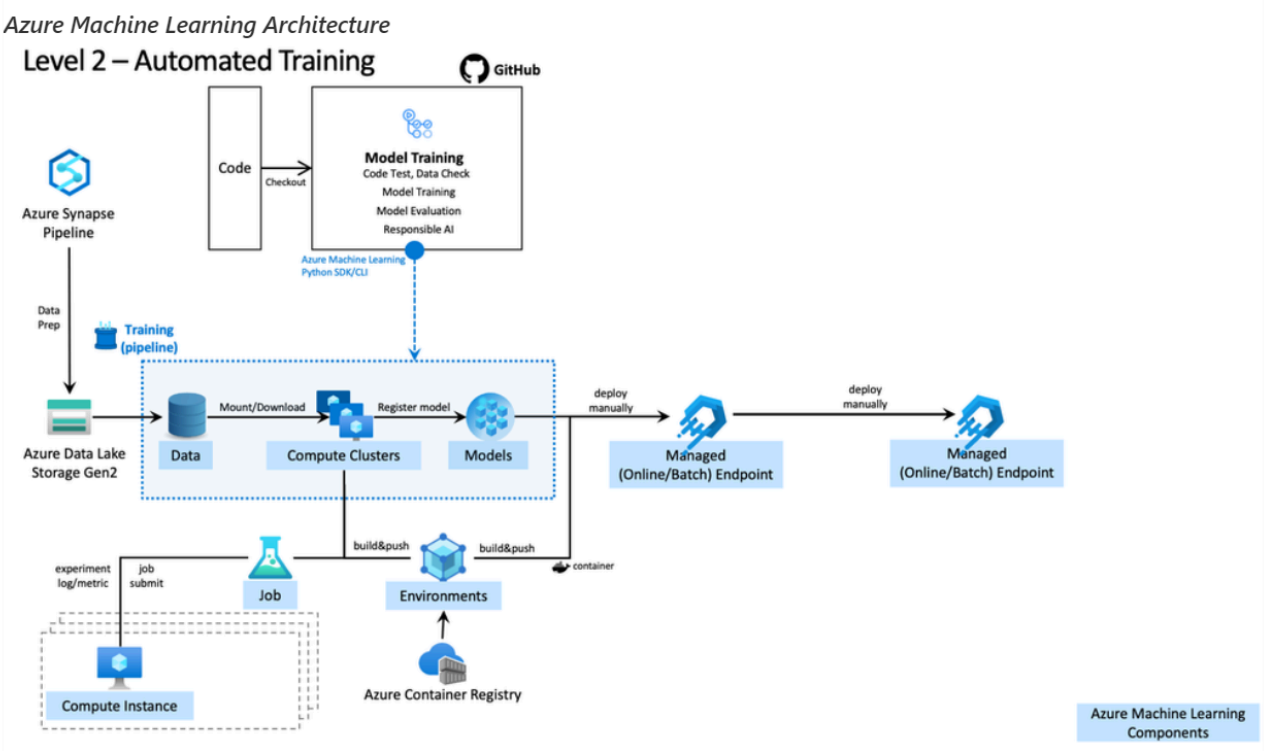

Figure 9 : Level 2 : Automated Training Azure MLOPS Architecture

The tables that follow identify the detailed characteristics for this level of process maturity.

People	Model Creation	Model Release	Application Integration
• Data scientists: Working directly with data engineers to convert experimentation code into repeatable scripts/jobs • Data engineers: Working with data scientists • Software engineers: siloed, receive model remotely from the other team members	• Data pipeline gathers data automatically • Compute managed • Experiment results tracked • Both training code and resulting models are version controlled	• Manual release • Scoring script is version controlled with tests • Release managed by Software engineering team	• Basic integration tests exist for the model • Heavily reliant on data scientist expertise to implement model • Application code has unit tests

Figure 10 : Detailed Characteristics For Level 2 Of Process Maturity

4.1.2 Physical Architecture

The physical architecture of Azure ML Studio involves using compute instances and virtual machines. Compute instances provide scalable, on-demand development environments tailored for machine learning tasks, while virtual machines offer versatile and customizable resources for training models, deploying applications, and managing workloads efficiently. This

setup ensures robust and flexible infrastructure support for all stages of the machine learning lifecycle.[15] [16]

Size	vCPU	Memory	Temp storage	Max Data Disks	Throughput	max NICs	Max Bandwidth
STANDARD_E4DS_V4	4	32	150	8	9000/125	2	2
STANDARD_D4S_V3	4	16	100	8	6000/93/46	2	2
Standard_D3_v2	4	14	200	16	12000/187/93	4	4

Table 1 : Characteristics of virtual machine sizes Azure

4.1.3 Software Environment

The software environment in Azure ML Studio is a meticulously curated setup to facilitate smooth machine learning development and deployment. It includes pre-configured virtual environments with popular machine learning frameworks such as TensorFlow, PyTorch, and scikit-learn. Azure ML Studio also integrates with Jupyter Notebooks for interactive data analysis and experimentation. Additionally, it supports Python and R programming languages, providing flexibility for various machine learning and data science tasks. Tools like Azure Data Factory and Azure Blob Storage are available for data ingestion and management, ensuring a comprehensive and efficient development ecosystem.

In Azure ML Studio, each model is deployed into production using a conda.yaml environment file that specifies all necessary dependencies. This file ensures a consistent and reproducible setup by listing the required libraries and their versions.

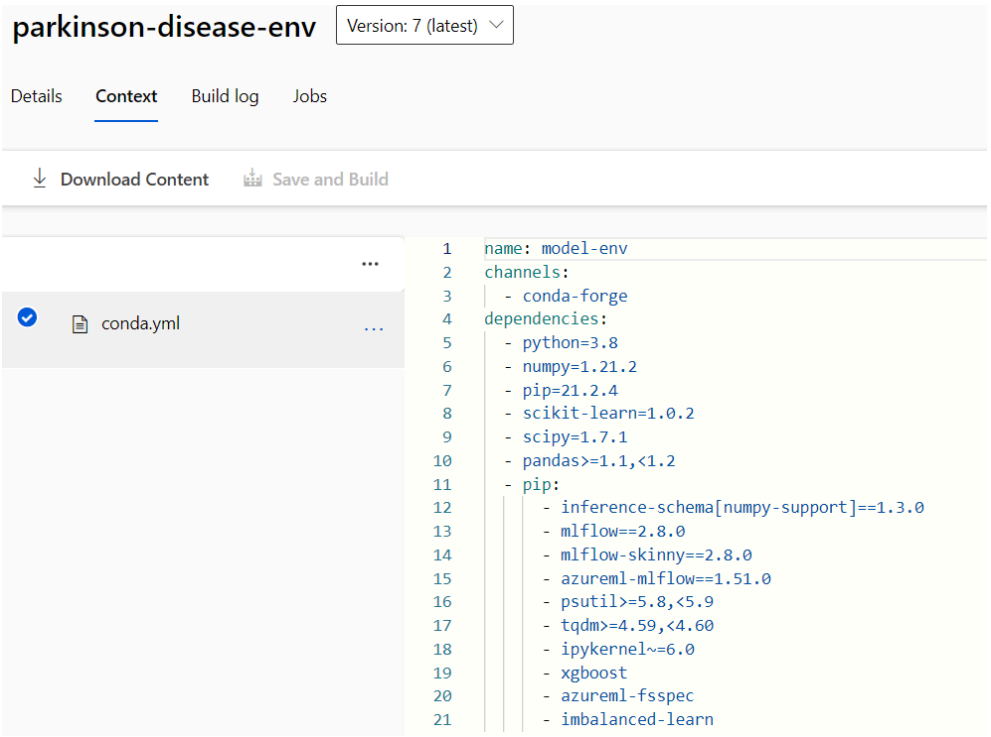

Figure 11 : Example of a conda.yaml file for parkinson disease environment

4.2. CRISP-DM Methodology

CRISP-DM is a widely accepted methodology in the data mining community that provides a structured approach to planning a data mining project. The CRISP-DM framework comprises of six main phases.

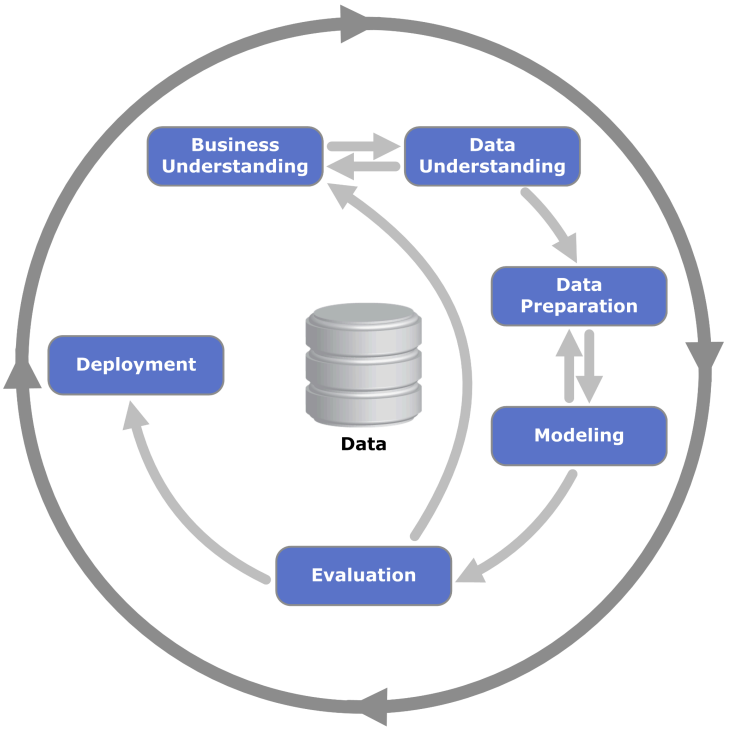

Figure 12 : CRISP-DM Diagram

The figure displays the six phases of the CRISP-DM methodology: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. [17]

4.2.1. Business Understanding

This is the initial phase which focuses on understanding the project objectives and requirements from a business perspective, and then converting this knowledge into a data mining problem definition.

4.2.2. Data Understanding

In this phase, an initial dataset is created based on the project requirements. Various data exploration techniques are applied to familiarize with the data, identify data quality problems, and discover first insights.

4.2.3. Data Preparation

The data preparation phase covers all activities needed to construct the final dataset which can be fed into the modeling tools. Tasks include table, record, and attribute selection as well as transformation and cleaning of data.

4.2.4. Modeling

In this phase, various modeling techniques are selected and applied. Often, the steps are performed more than once and there are several models to consider.

4.2.5. Evaluation

Before proceeding to final deployment, the model needs to be evaluated thoroughly. This phase helps to identify if the model meets the business objectives and to determine any shortcomings.

Various metrics were employed to evaluate and optimize the performance of the models.

- **Accuracy** is the most straightforward metric, quantifying the overall correctness of the model's predictions. The expression is as follows, where T indicates True, F indicates False, N is Negatives and P is Positives:

$$\text{accuracy} = \frac{TP+TN}{TP+FN+TN+FP}$$

- **Precision** evaluates the model's ability to correctly identify positive instances out of all the instances it has predicted as positive.

$$\text{precision} = \frac{TP}{TP+FP}$$

- **Recall**, also known as sensitivity or true positive rate, quantifies the model's ability to identify all positive instances from the actual positives.

$$\text{recall} = \frac{TP}{TP+FN}$$

- **F1 score** The score is the harmonic mean of precision and recall, offering a balanced measure when both are important.

$$\text{F1 score} = \frac{2 * precision * recall}{precision + recall}$$

- **Loss function** For this project, we used Categorical Crossentropy as a loss function for multiclass classification problems.

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

Figure 13 : Loss expression

The figure 13 shows that the loss function calculates the error rate between the predicted and actual distributions. Its value can range from 0 to infinity, with 0 indicating that the predicted probabilities perfectly match the actual distribution**.**

4.2.6. Deployment

The knowledge discovered during the modeling phase is organized and presented in a way that the customer can use it. It can be as simple as a report, or as complex as a repeatable data mining process.

- **Azure Machine Learning Studio**

Azure Machine Learning is a cloud service for accelerating and managing the machine learning (ML) project lifecycle. ML professionals, data scientists, and engineers can use it in their day-to-day workflows to train and deploy models and manage machine learning operations (MLOps). [18]

- **MLflow**

MLflow is an open-source platform, purpose-built to assist machine learning practitioners and teams in handling the complexities of the machine learning process. MLflow focuses on the full lifecycle for machine learning projects, ensuring that each phase is manageable, traceable, and reproducible.[19]

- **Datastore**

An Azure Machine Learning datastore serves as a *reference* to an *existing* Azure storage account. An Azure Machine Learning datastore offers these benefits:

  - A common, easy-to-use API that interacts with different storage types (Blob/Files/ADLS).
  - Easier discovery of useful datastores in team operations.
  - For credential-based access (service principal/SAS/key), Azure Machine Learning datastore secures connection information. This way, you won't need to place that information in your scripts. [20]

- **Model Registry**

Model registration allows you to store and version your models in the Azure cloud, in your workspace. The model registry helps you organize and keep track of your trained models [21]

- **Environments**

Azure Machine Learning environments are an encapsulation of the environment where your machine learning training happens. They specify the Python packages, and software settings around your training and scoring scripts. The environments are managed and versioned entities within your Machine Learning workspace that enable reproducible, auditable, and portable machine learning workflows across various compute targets.[22]

- **Endpoints**

An endpoint, in this context, is an HTTPS path that provides an interface for clients to send requests (input data) to a trained model and receive the inferencing (scoring) results from the model. An endpoint provides:

- Authentication using "key or token" based auth
- TLS(SSL) termination
- A stable scoring URI (endpoint-name.region.inference.ml.azure.com)

A deployment is a set of resources required for hosting the model that does the actual inferencing.

A single endpoint can contain multiple deployments. Endpoints and deployments are independent Azure Resource Manager resources that appear in the Azure portal.

Azure Machine Learning allows you to implement online endpoints for real-time inferencing on client data, and batch endpoints for inferencing on large volumes of data over a period of time.

Therefore, because of its systematic and flexible approach, which fits well with the iterative nature of the project's pipelines, we used the CRISP-DM technique for our MLOps project. From understanding the business challenge to deployment, CRISP-DM led us through important stages. Its adaptability supports the MLOps' dynamic character, making it a strong strategy for our project.[23]

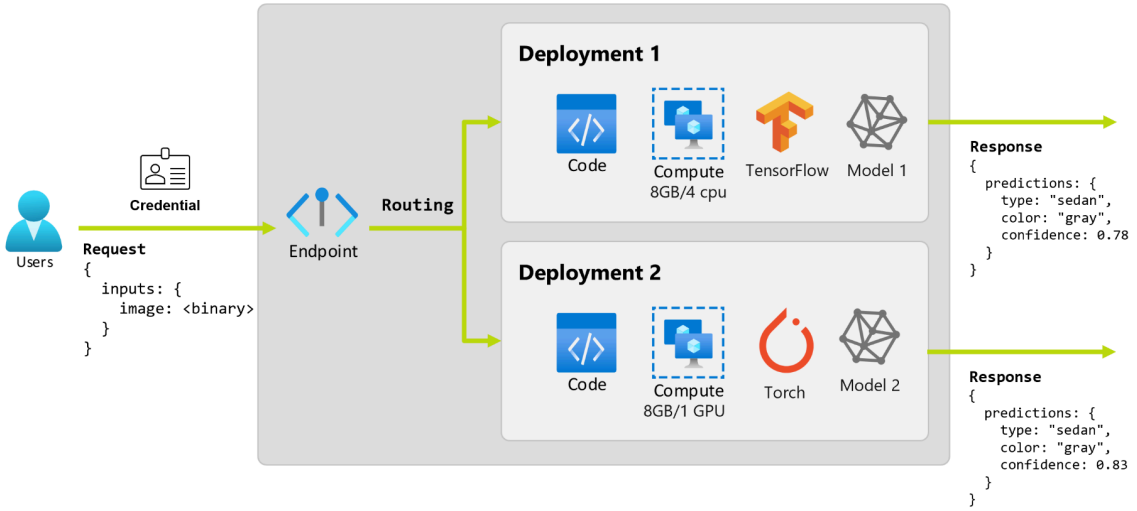

Figure 14: Endpoints Architecture

4.3. Different Components of the architecture

We streamlined the creation, deployment, and monitoring phases of the lifecycle of our machine learning models for this project using MLOps. Our models' dependability and robustness are increased by MLOps' reliable, reproducible, and effective procedures. Additionally, it guarantees that as our models continue to learn from and adapt to new data, they will remain effective and pertinent.

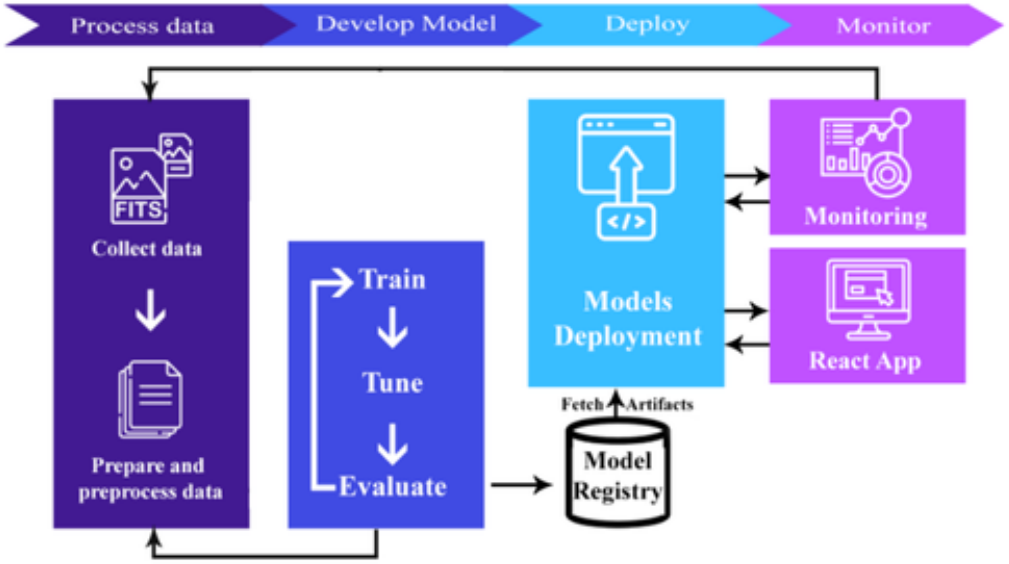

Figure 15 : MLOps Architecture overview

Figure 15 – MLOps Architecture overview The figure displays the four phases of the MLOps pipeline, data processing, model development, deployment and monitoring, highlighting the crucial processes in our end-to-end machine learning pipeline.

- **Process Data:** Collect and preprocess medical data, ensuring its quality and readiness for training.

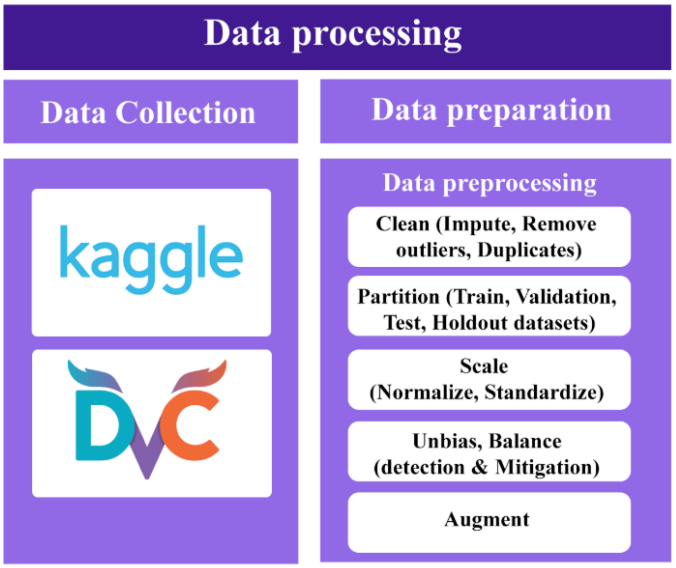

Figure 16 : – Data processing phase

The figure above (figure 16) displays the two steps of data processing phase: the data collection and the data preparation.

- **Develop Model**: Train, tune, and evaluate each model to optimize their performance.
- **Store Models in Model Registry**: Store the trained models' artifacts in the Model Registry for easy versioning and management.
- **Deploy**: Deploy the models into production, exposing them through a React application interface for user interaction.

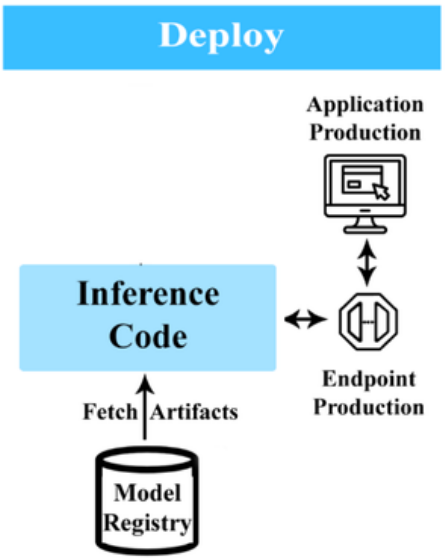

Figure 17 : Deployment phase

Figure 17 showcases the deployment phase of our pipeline. We use the combined power of the Feature Store and Model Registry to ensure seamless integration and the effective deployment of our models into the production environment.

- **Monitor**: Continuously monitor model performance metrics, system-level metrics, and detect drift to ensure ongoing reliability.

This architecture highlights the seamless integration of components for accurate predictions and efficient management of the machine learning lifecycle.

Conclusion

In this chapter, we explored the critical components of our healthcare application, focusing on technological choices, including the logical and physical architectures and software environment. We detailed the CRISP-DM methodology and its phases, covering business understanding, data preparation, modeling, evaluation, and deployment. We also described the deployment process using Azure Machine Learning Studio and outlined the key aspects of our MLOps framework, emphasizing data processing, model development, deployment, and continuous monitoring. This comprehensive approach ensures robust and scalable solutions for accurate medical predictions.

Deep Learning And Machine Learning Models Design

Introduction

In this final chapter, we delve into the specific models and architectures employed for disease detection and prediction in our healthcare application. We begin with Parkinson's disease detection, exploring the XGBoost, Random Forest, and K Neighbors architectures, followed by a metrics comparison to select the best model. We then examine the chosen model's training, validation, and testing results. Next, we cover Alzheimer's stages prediction using InceptionV3 and VGG19 architectures, and Brain Tumor stages prediction with Sequential CNN and EfficientNetB0 architectures, again comparing metrics and discussing training, validation, and testing outcomes. Lastly, we address Skin Cancer detection using Support Vector Machine and EfficientNetB3 architecture, completing the discussion with model comparisons and results.

5.1. Parkinson Disease Detection

5.1.1. XGboost Architecture

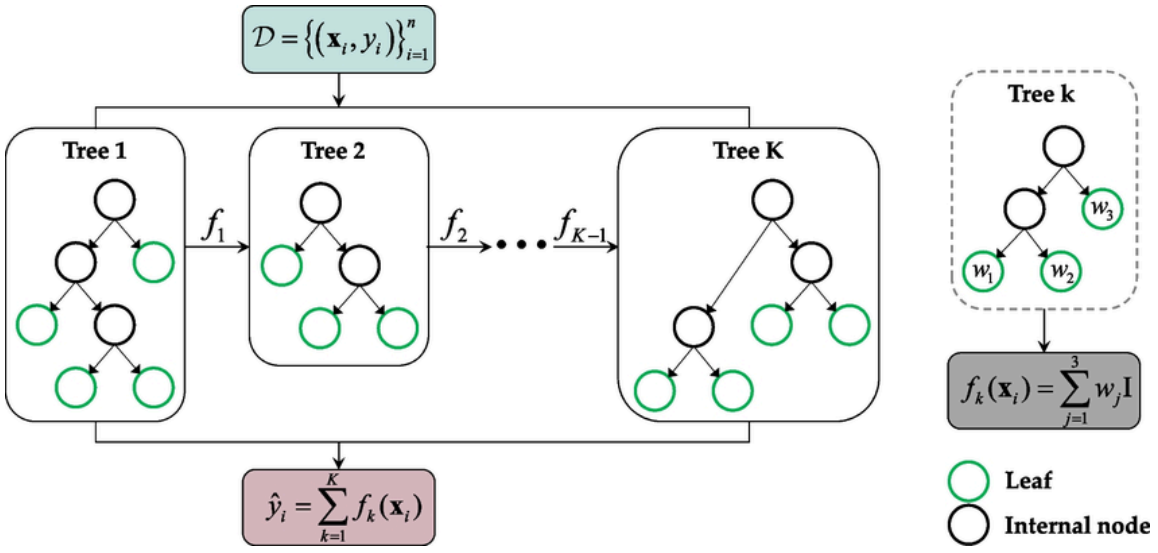

Figure 18 : XGboost Architecture

The figure depicts the architecture of an XGBoost model, a popular implementation of gradient boosting. [24]

Gradient Boosting

Gradient Boosting uses differentiable function losses from the weak learners to generalize. At each boosting stage, the learners are used to minimize the loss function given the current model. Boosting algorithms can be used either for classification or regression.

$$F_{x_t+1} = F_{x_t} + \epsilon_{x_t} \frac{\partial F}{\partial x}(x_t)$$

To iterate over the model and find the optimal definition we need to express the whole formula as a sequence and find an effective function that will converge to the minimum of the function. This function will serve as an error measure to help us decrease the loss and keep the performance over time. The sequence converges to the minimum of the function . This particular notation defines the error function that applies when evaluating a gradient boosting regressor.

$$f(x, \theta) = \sum l(F((X_i, \theta), y_i))$$

5.1.2. Random Forest Architecture

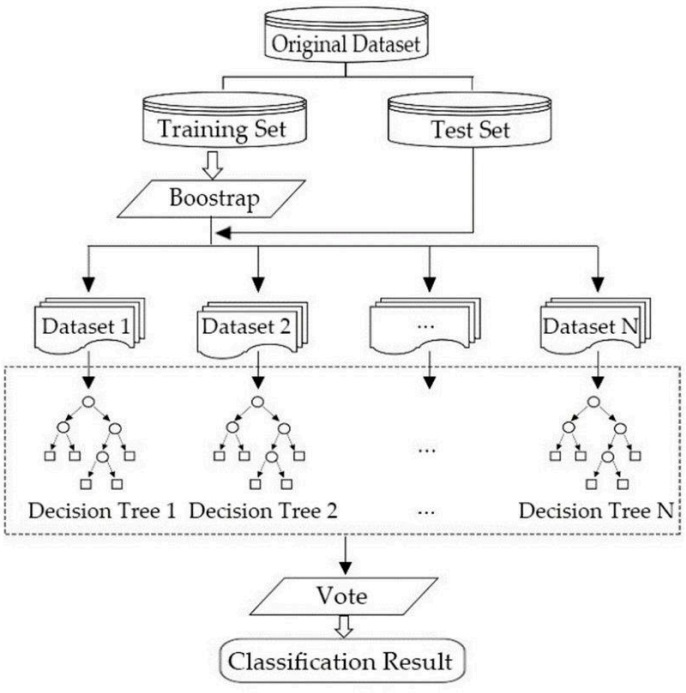

Figure 19 : Random Forest Architecture

This figure illustrates the architecture of a Random Forest model, a popular ensemble learning method used for classification and regression tasks. Here is a brief explanation of the architecture:

1. Original Dataset:

- The original dataset is split into a training set and a test set. The training set is used to build the model, while the test set is used to evaluate its performance.

2. Bootstrap Sampling:

- The training set is used to create multiple bootstrap samples (Dataset 1, Dataset 2, ..., Dataset N). Each bootstrap sample is generated by randomly sampling the training set with replacement, ensuring diversity among the datasets.

3. Building Decision Trees:

- For each bootstrap sample, a decision tree is constructed. These decision trees are typically built to their maximum depth without pruning. Each tree is trained independently, learning different aspects of the data.

4. Decision Tree Structure:

- Each decision tree is composed of internal nodes (circles) that represent decision points based on feature values, and leaf nodes (squares) that represent the output or class labels.

5. Voting Mechanism:

- Once all decision trees are built, they are used to make predictions on new data. Each tree provides a classification result, and the final output is determined by a majority vote among all the trees. For regression tasks, the average of the predictions is used.

6. Classification Result:

- The aggregated votes from all the decision trees yield the final classification result for the input data. This ensemble approach helps in improving the accuracy

and robustness of the model by reducing overfitting and variance.[25]

5.1.3. K Nearest Neighbors Architecture

KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.[26]

As an example, consider the following table of data points containing two features:

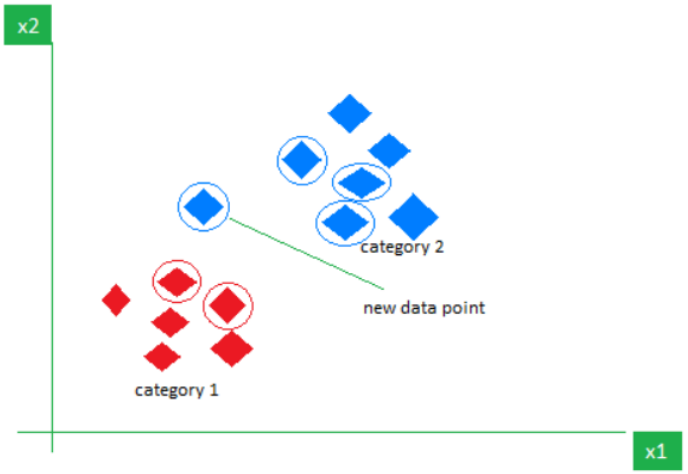

Figure 20 : KNN Algorithm working visualization

5.1.4. Metrics Comparison

During Training:

Metric	Accuracy	Cross Validation	Precision	Recall	F1 Score
KNN	0.968	0.920	0.971	1	0.973
XGboost	1	0.920	1	1	1
Random Forest	1	0.909	1	1	1

Table 2 – Results comparison: KNN vs XGboost vs Random Forest during training

During Testing:

Metric	Accuracy	Cross Validation	Precision	Recall	F1 Score
KNN	0.893	0.806	0.878	0.966	0.920
XGboost	1	0.913	1	1	1
Random Forest	1	0.893	1	1	1

Table 3 – Results comparison: KNN vs XGboost vs Random Forest during testing

Achieving a training accuracy of 1.0, as observed previously, suggests that the model has perfectly learned the training data. However, this does not guarantee that the model will generalize well to unseen data. Similarly, a testing accuracy of 1.0 can indicate perfect performance on the testing data, but it is crucial to assess whether this level of performance is realistic and whether the model has genuinely learned meaningful patterns from the data.

In our case, overfitting is likely because our data is unbalanced. To address this, we will focus on hyperparameter tuning to reduce overfitting and improve the model's ability to generalize to new, unseen data.

Random Forest

```
Evaluation Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
cross_validation_score: 0.8733333333333334

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      1.00      1.00        30

    accuracy                           1.00        47
   macro avg       1.00      1.00      1.00        47
weighted avg       1.00      1.00      1.00        47
```

Figure 21 : Classification Report for Random Forest after hyper parameters tuning

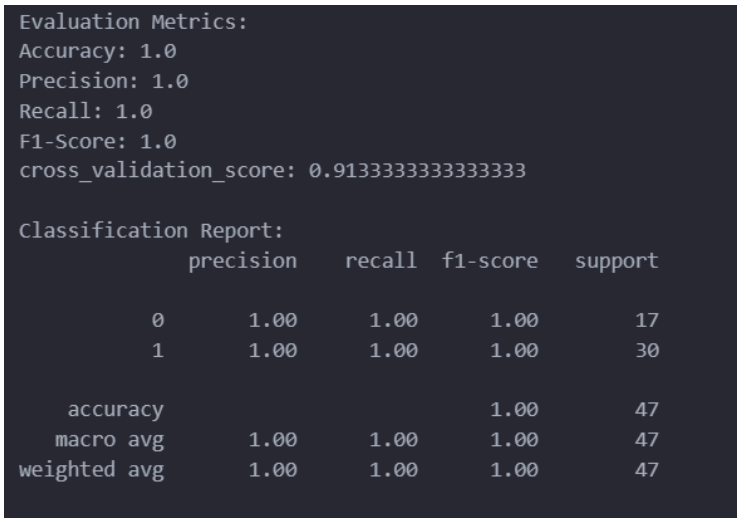

Figure 22 : Classification Report for XGBoost after hyper parameters tuning

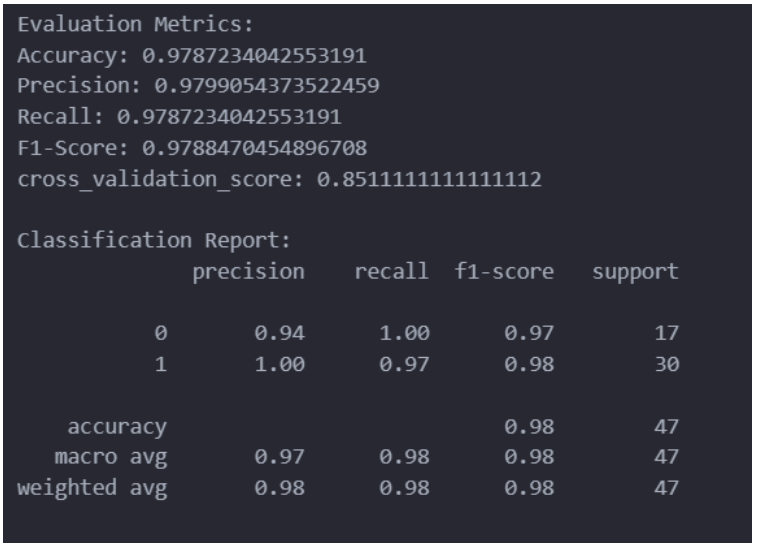

Figure 23 : Classification Report for KNN after hyper parameters tuning

5.1.5. Chosen Model

The three above figures show that the models demonstrate high accuracy and F1-scores, indicating strong performance. However, there are some differences to consider:

* Random Forest & XGboost achieves higher metrics yet we have higher risks of overfitting.

* K Nearest Neighbors achieves good metric less than the other two yet less risk of overfitting

Given the performance of both models, we opt for K Nearest Neighbors Model since it achieves a good accuracy 97.87% and is less prone to overfitting.

5.1.6. Chosen Model : Training and Validation Results

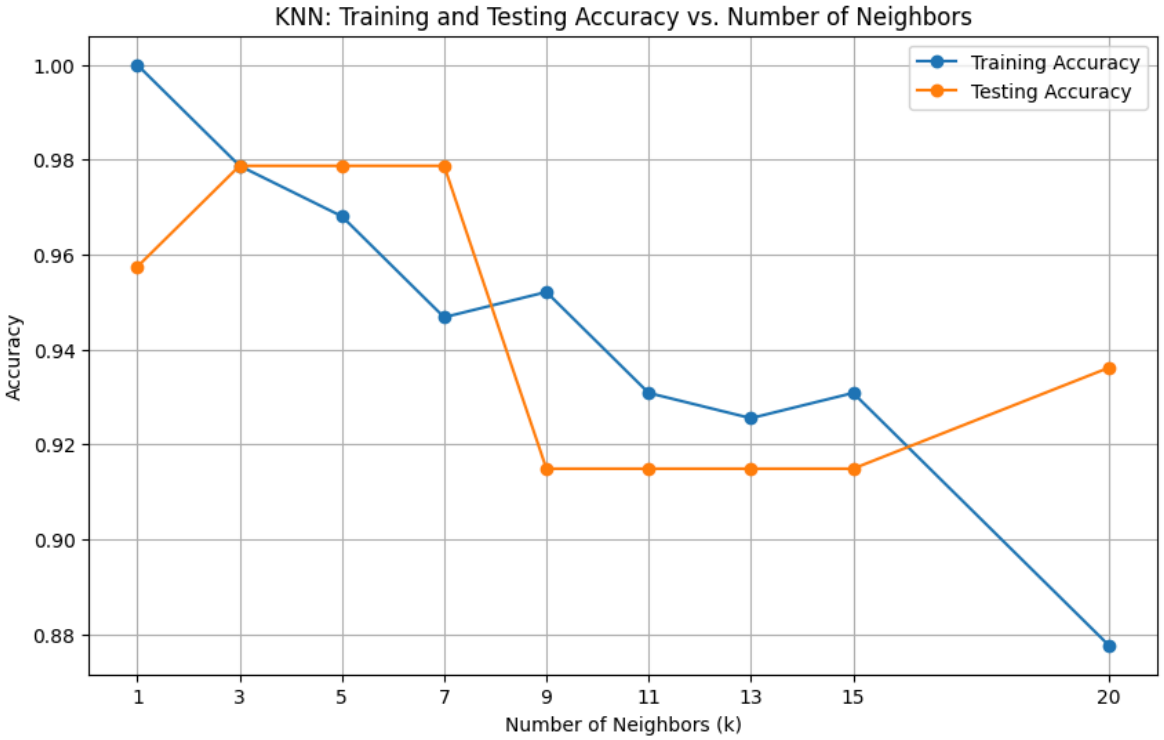

Figure 24 : KNN Training and Validation Accuracy Results

This plot displays the training and testing accuracy of a k-Nearest Neighbors (KNN) classifier as a function of the number of neighbors (k). Here are some observations and interpretations based on the graph:

- **Training Accuracy:**

- The training accuracy starts at 1.0 for (k=1), indicating perfect fit (potential overfitting) when k is very small.

- As k increases, the training accuracy gradually decreases, showing a decline in model complexity and less overfitting.

- **Testing Accuracy:**

- The testing accuracy is slightly lower than the training accuracy at small k values but tends to be more stable compared to the training accuracy as k increases.

- For small values of k (specifically k=1), the model seems to overfit, shown by a drop in testing accuracy compared to training accuracy.

- There's a notable decrease in testing accuracy around k=7 to k=9, followed by a relatively flat trend indicating stability but lower accuracy.

- **Optimal k:**

- The plot suggests that the optimal k (balancing both training and testing accuracy) lies somewhere between k=3 to k=7. This range maintains high testing accuracy without significant overfitting.

- Beyond k=7, there is a drop in testing accuracy, and the testing accuracy remains quite low, indicating underfitting.

Choosing k between 3 and 7 seems to provide the best balance between training and testing accuracy, making it a reasonable choice for this specific dataset.

5.1.7. Testing Results

Upon entering a converted voice recording into a CSV file for an individual without Parkinson's disease, the following result is retrieved from the Flask backend of the application:

Figure 25 : Parkinson Detection Model testing

5.2. Alzheimer's Stages Prediction

5.2.1. InceptionV3 Architecture

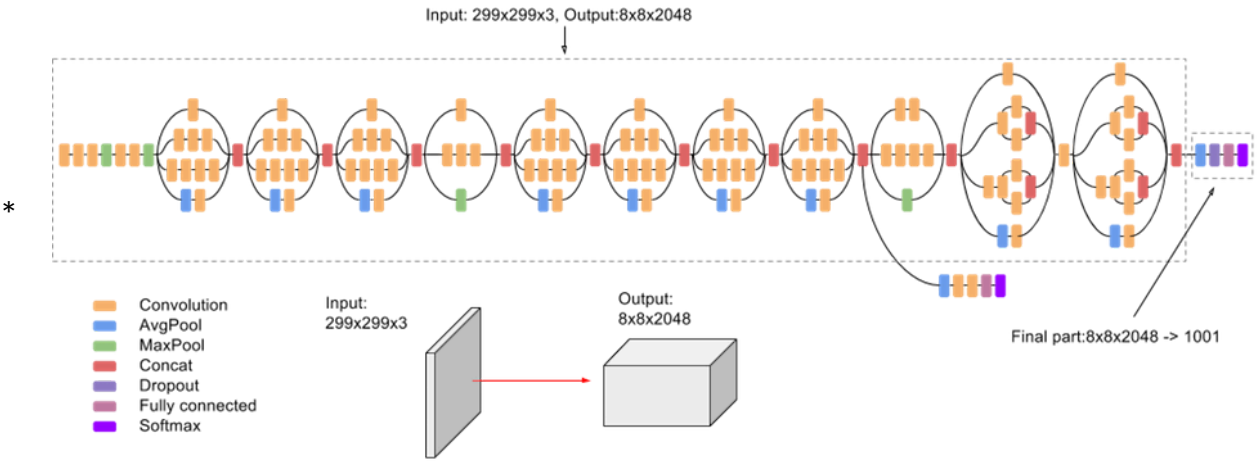

Figure 26 : InceptionV3 Architecture

Inception-v3 is a convolutional neural network architecture from the Inception family that makes several improvements including using Label Smoothing, Factorized 7 x 7 convolutions, and the use of an auxiliary classifier to propagate label information lower down the network (along with the use of batch normalization for layers in the side head).[27]

5.2.2. VGG19 Architecture

VGG19, a variant of Convolutional Neural Networks, is characterized by its deep architecture consisting of 19 layers, including 16 convolutional layers and 3 fully connected layers, which contributes to its strong image classification performance.

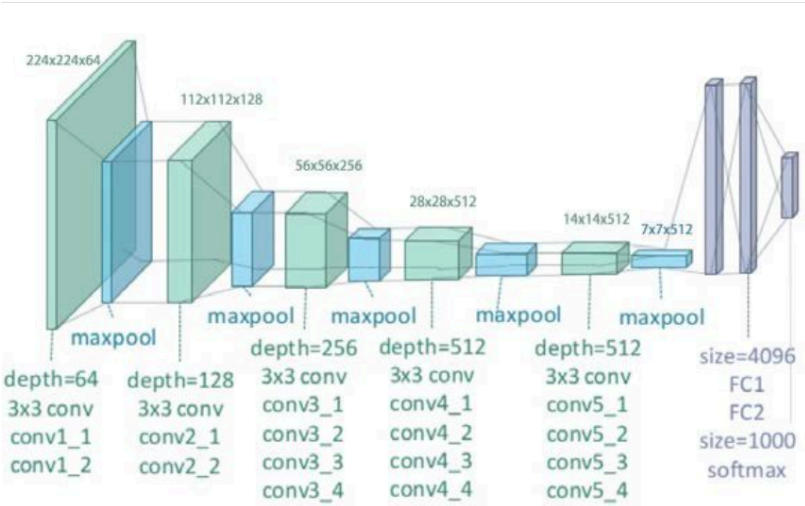

Figure 27 : VGG19 Architecture image

The figure above (figure 26) presents the structure of the VGG19 model. The model begins with an input layer, where raw image data is initially processed. This is followed by a series of 16 convolutional layers that progressively extract higher-level features from the input image. The architecture culminates in three fully connected layers which combine these high level features to make the final classification decision.

The following definitions can explain the key features of this architecture.

(a) Input Layer: Accepts images with a fixed size, typically 224x224 pixels.

(b) Convolutional Layers: 16 convolutional layers with 3x3 filters and ReLU activation functions, organized into five blocks. Each block contains two to four convolutional layers, followed by a max-pooling layer.

(c) Fully Connected Layers: Three fully connected layers, with the first two having 4096 neurons each and ReLU activation functions. The last fully connected layer has as many neurons as the number of classes, with a softmax activation function for multi-class classification. [28]

5.2.3. Metrics Comparison

During training:

InceptionV3

Epochs/Metric	Accuracy	Loss	Val Accuracy	Val Loss
0	0.419	1.447	0.622	0.809
4	0.686	0.683	0.739	0.552
67	0.954	0.136	0.959	0.117
71	0.961	0.120	0.956	0.121

Table 4 – Results comparison: InceptionV3 during training

VGG19

Epochs/Metric	Accuracy	Loss	Val Accuracy	Val Loss
0	0.7	0.710	0.798	0.483
5	0.932	0.205	0.931	0.217
10	0.973	0.106	0.946	0.156
24	0.994	0.038	0.966	0.097

Table 5 – Results comparison: VGG19 during training

During testing:

Metric	Accuracy	Loss
InceptionV3	0.945	0.154
VGG19	0.966	0.092

Table 6 – Results comparison: VGG19 vs InceptionV3

Analysis and Conclusion

- Training Accuracy and Loss: VGG19 reaches a higher training accuracy (0.994) with a lower training loss (0.038) compared to InceptionV3's best training accuracy (0.961) and loss (0.120). This indicates that VGG19 fits the training data better.
- Validation Accuracy and Loss: VGG19 also shows a higher validation accuracy (0.966) and lower validation loss (0.097) at its best epoch compared to InceptionV3's best validation accuracy (0.959) and loss (0.117). This suggests better generalization on the validation set for VGG19.
- Testing Performance: During testing, VGG19 outperforms InceptionV3 with a higher accuracy (0.966 vs. 0.945) and a lower loss (0.092 vs. 0.154).

Given the superior performance of VGG19 in both training and testing phases, including higher accuracy and lower loss values, VGG19 is the adequate model to choose for this task. It not only fits the training data well but also generalizes better to unseen data.

5.2.4. Chosen Model : Training and Validation Results

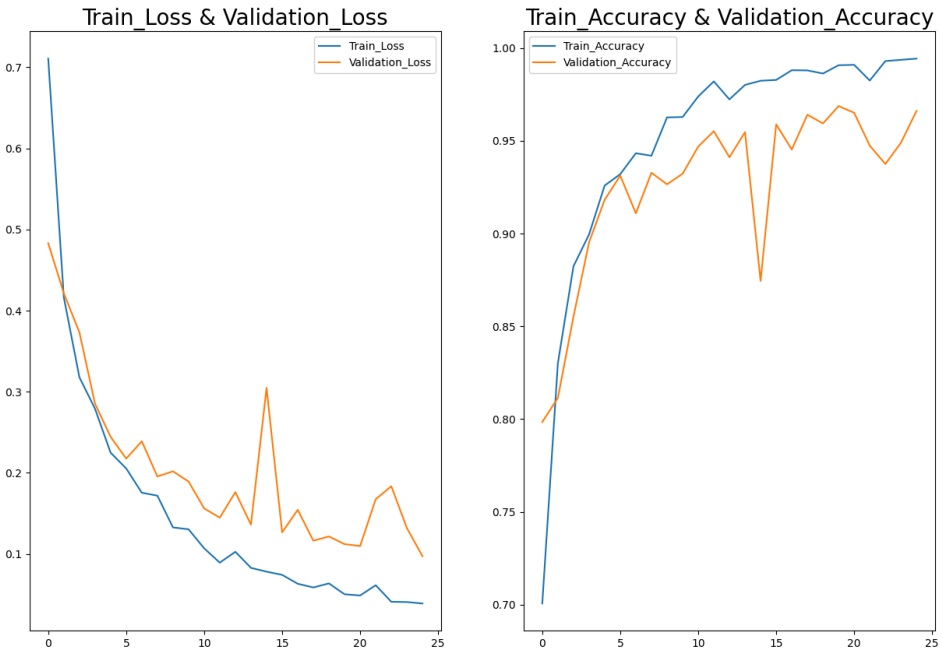

Figure 28 : VGG19 Training and Validation Accuracy Results

The provided graphs display the training and validation performance of the VGG19 model over 25 epochs.

Overall, the VGG19 model demonstrates strong training and validation performance, with high accuracy and low loss metrics. While there is some evidence of overfitting, the model generalizes well to the validation data, making it a robust choice. Given the consistent improvement over epochs, further training or fine-tuning (e.g., regularization, dropout) might help mitigate overfitting and improve validation performance further.

5.2.5. Testing Results

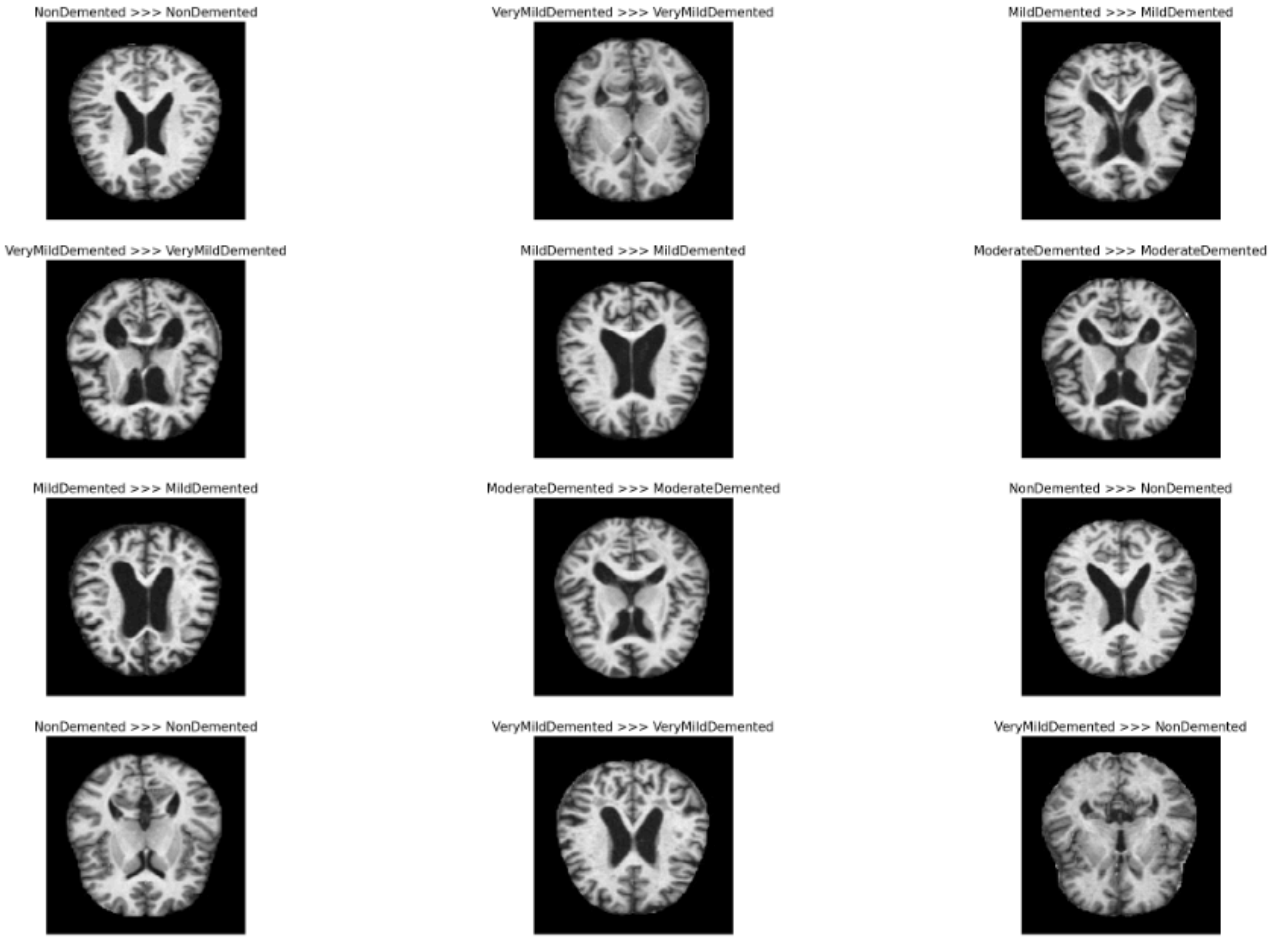

Figure 29 : Alzheimer's Stages Prediction Model Testing

The figure shows an example of the model prediction the stage of Alzheimer's in multiple patients from MRI scans.

5.3. Brain Tumor Stages Prediction

5.3.1. Sequential CNN Architecture

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. The cnn architecture uses a special technique called Convolution instead of relying solely on matrix multiplications like traditional neural networks. Convolutional networks use convolution, which in mathematics is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.[29]

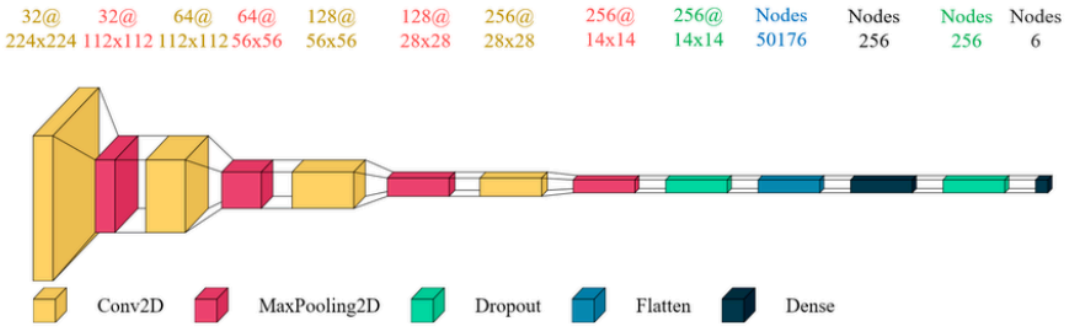

Figure 30 : CNN Architecture

The figure shows that the Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.

The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

5.3.2. EfficientNetB0 Architecture

EfficientNetB0 is a state-of-the-art convolutional neural network architecture that is part of the EfficientNet family, which was developed to achieve high performance while being computationally efficient. [30]

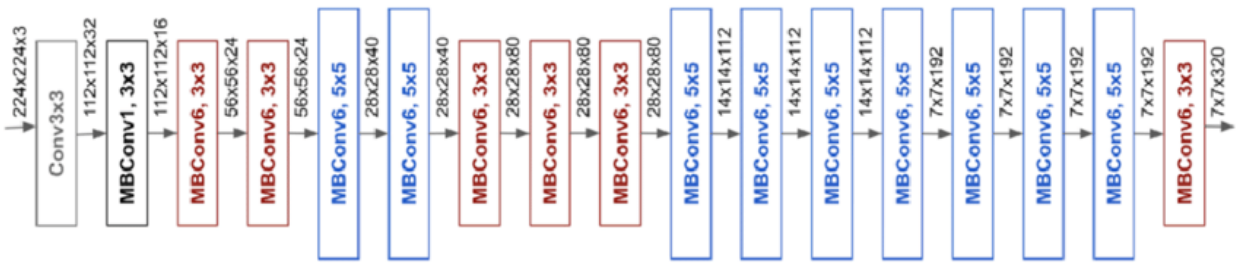

Figure 31 : EfficientNetB0 baseline model architecture

5.3.3. Metrics Comparison

CNN

Classification report:

	precision	recall	f1-score	support
0	0.84	0.92	0.88	97
1	0.88	0.90	0.89	48
2	0.90	0.76	0.82	101
3	0.94	1.00	0.97	81
accuracy			0.89	327
macro avg	0.89	0.89	0.89	327
weighted avg	0.89	0.89	0.88	327

EfficientNetB0

Classification report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	97
1	0.98	0.98	0.98	48
2	1.00	0.97	0.98	101
3	0.96	1.00	0.98	81
accuracy			0.98	327
macro avg	0.98	0.98	0.98	327
weighted avg	0.99	0.98	0.98	327

5.3.4. Chosen Model : Training and Validation Results

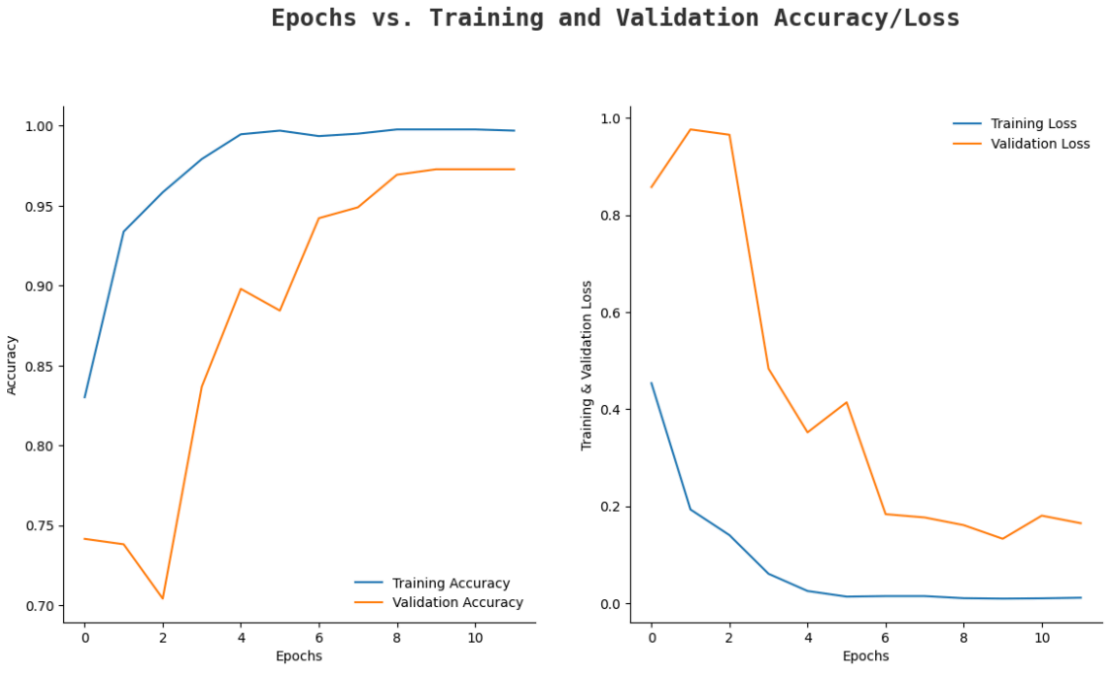

Figure 32 : EfficientNetB0 Training and Validation Accuracy/Loss Results

The provided graphs depict the training and validation performance metrics of a neural network model over 12 epochs.

Overall, the model shows robust training and validation performance. While there is a slight indication of overfitting, the high validation accuracy and decreasing validation loss suggest that the model generalizes well to unseen data. To mitigate overfitting further, techniques such as dropout, early stopping, or data augmentation could be considered. Nonetheless, the current performance metrics indicate a well-trained model that is capable of making accurate predictions on new data.

5.3.5. Testing Results

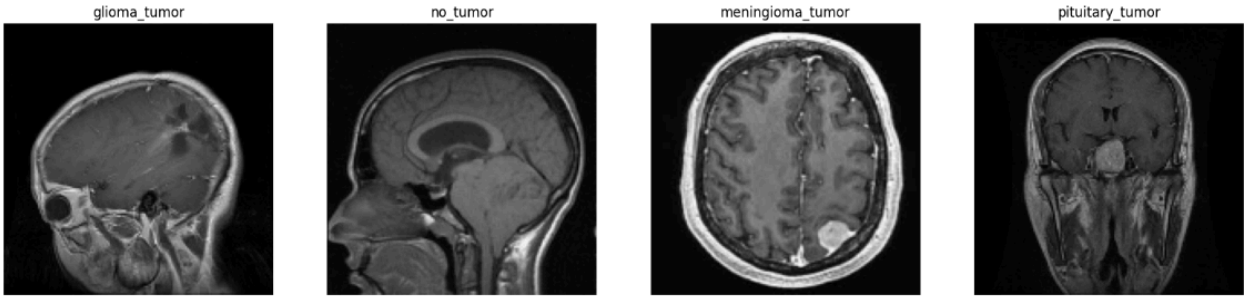

Figure 33 : Brain tumor Types classification Model Testing

The figure shows an example of the model predicting the types of brain tumor in multiple patients from MRI scans.

5.4. Skin Cancer Detection

5.4.1. Support Vector Machine Architecture

SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees. It is known for its kernel trick to handle nonlinear input spaces.SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplanes in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.[31]

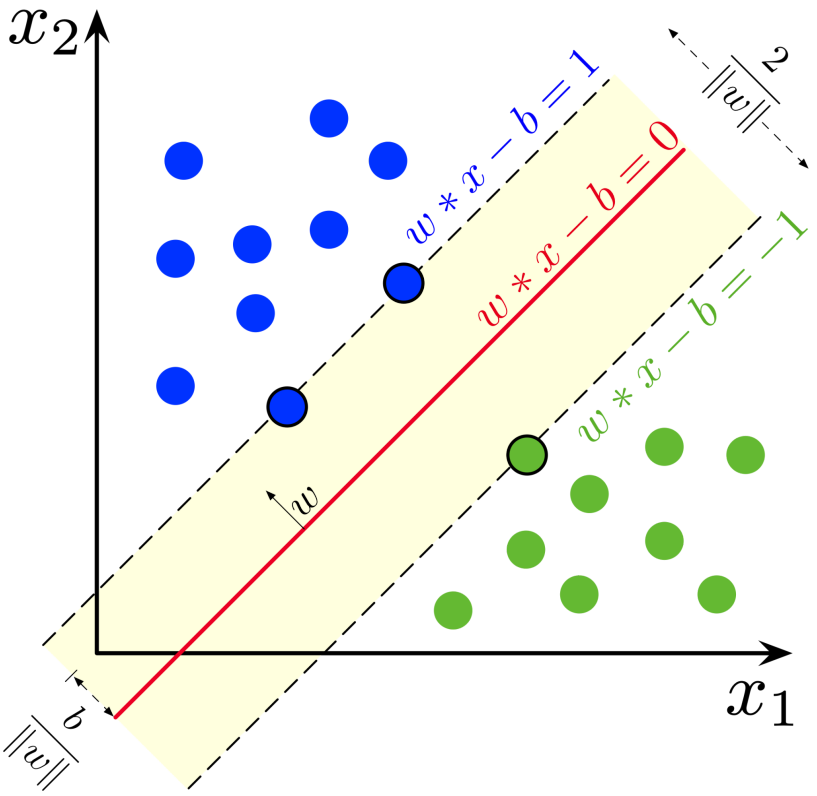

Figure 34 : SVM Architecture

The figure 33 displays the SVM architecture, showing the Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

5.4.2. EfficientNetB3 Architecture

EfficientNet-B3 is part of the EfficientNet family of models, which are known for their balance of efficiency and performance in image classification tasks. EfficientNet-B3 builds on the architecture of EfficientNet-B0 by applying a compound scaling method that uniformly scales the network's width, depth, and resolution, resulting in improved accuracy and efficiency. Specifically, EfficientNet-B3 has more layers and larger filter sizes compared to B0. This scaling results in a network that can capture more detailed features from the input images while maintaining computational efficiency. EfficientNet-B3 retains key architectural features such as the use of Mobile Inverted Bottleneck Convolution (MBConv) layers and the Swish activation function, which contribute to its high performance on image classification benchmarks. [32]

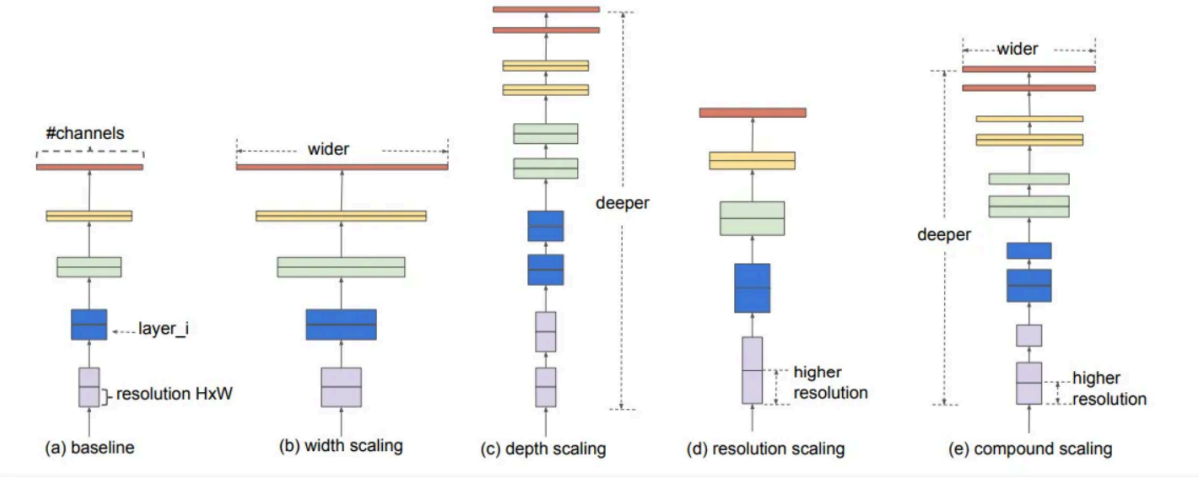

Figure 35 : EfficientNetB3 Architecture

The figure 34 displays the EfficientNetB3 architecture, showing the Model Scaling.

(a) is a baseline network example;

(b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution.

(e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

5.4.3. Metrics Comparison

- **During training**

SVM Accuracy: 87.2%

EfficientNetB3

Epochs/Metric	Accuracy	Loss	Val Accuracy	Val Loss
1	0.76766	6.977	0.6212	7.21075
5	0.94784	2.364	0.871	2.26381
10	0.99004	0.990	0.99004	0.685
15	0.99004	0.402	0.88258	0.2086

Table 7 – Results comparison: EfficientNetB3 during training

Train Loss: 0.208

Train Accuracy: 0.99

- **During testing**

SVM Accuracy: 83.4%

EfficientNetB3:

Test Loss: 0.445

Test Accuracy: 0.909

5.4.4. Chosen Model : Training and Validation Results

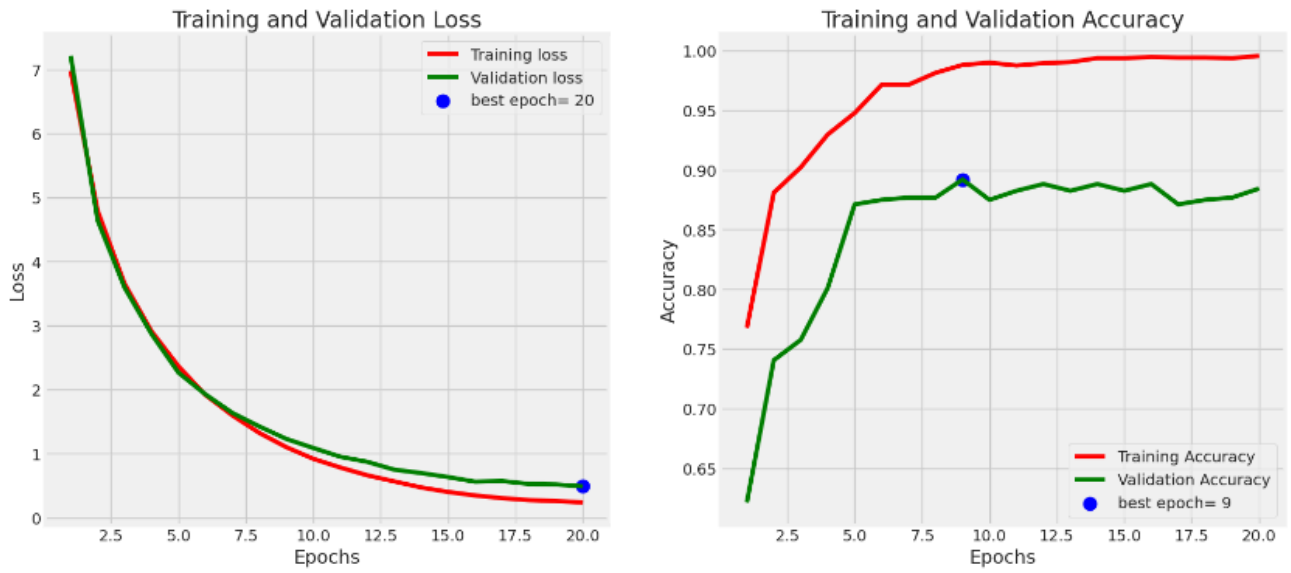

Figure 36 : EfficientNetB3 Training and Validation Results

The graphs illustrate the training and validation loss (left) and accuracy (right) over 20 epochs. The training loss consistently decreases, indicating effective learning, while the validation loss also declines, suggesting good generalization. The training accuracy reaches nearly 100%, but the validation accuracy plateaus around 90%, with the best epoch identified at 9. This suggests potential overfitting, where the model performs exceptionally well on training data but less so on validation data.

5.4.5. Testing Results

Figure 37 : Skin Cancer Prediction Model testing

Upon entering an image of a skin rash for an individual with a malignant tumor, the following result is retrieved from the Flask backend of the application:

Conclusion:

This chapter presents the design and implementation of deep learning and machine learning models for detecting and predicting Parkinson's disease, Alzheimer's stages, brain tumor stages, and skin cancer. Each section provides detailed architectures, metric comparisons, and performance evaluations for models like XGBoost, Random Forest, K Neighbors, InceptionV3, VGG19, Sequential CNN, EfficientNetB0, and EfficientNetB3. The chapter concludes with the selection of the best-performing models based on training, validation, and testing results, highlighting their effectiveness in medical diagnostics.

General Conclusion and Perspectives

In our project, we aimed to utilize machine learning and deep learning techniques to tackle the challenges in the healthcare domain. We employed the CRISP-DM methodology for its systematic and well-defined approach, guiding us from understanding business requirements to deploying the model. This methodology facilitated the successful development of our system using VGG19, EfficientNetB0, EfficientNetB3, and K Nearest Neighbors, all seamlessly integrated into a responsive, user-friendly web application built with React.js and Flask.

Our models efficiently handled multiple datasets, minimizing human error and saving valuable time. The results included an accuracy of 0.909 for skin cancer detection, 0.98 for brain tumor type classification, 0.966 for Alzheimer's stage prediction, and 0.98 for Parkinson's disease detection. Although our datasets presented issues such as imbalance, we addressed these challenges successfully with techniques like SMOTE, demonstrating the efficacy of our approach.

We faced other challenges including the deployment of deep learning models, compatibility issues due to different TensorFlow and Keras versions, and selecting the appropriate compute resources without knowing the expected traffic. Ensuring a seamless experience for our clients while staying within budget was also a key consideration.

Our project featured the development of an interactive web interface aimed at making advanced models accessible to medical staff and healthcare researchers. This platform allows users to upload MRI scans to predict skin cancer, brain tumors, and Alzheimer's stages, as well as voice recordings to detect Parkinson's disease. Additionally, we developed an advanced AI chatbot specialized in healthcare to facilitate research and decision-making for medical staff.

All four models were deployed in Azure Machine Learning Studio with MLOps principles in mind, making each model accessible via its own endpoint in the cloud.

Looking ahead, we plan to deploy the entire website in Azure Web Service containers to enhance accessibility and scalability. We also aspire to achieve a more powerful MLOps architecture, such as Level 4: Full MLOps Automated Retraining in Azure, to fully automate all tasks.

Bibliography

[1] IBM watson
[2] Exploratory data analysis
[3] Machine learning
[4] Deep learning
[5] Francois Chollet. Deep Learning with Python. (2017). 7, 8
[6] Natural language processing
[7] Flask
[8] React
[9] Azure Cloud Provider
[10] Parkinson Disease Prediction
[11] Alzheimer's Stages Classification Kaggle Dataset
[12] Skin Cancer: Malignant vs Benign Kaggle Dataset
[13] Brain Tumor Types Classification Kaggle Dataset
[14] Level 2 MLops azure architecture
[15] Azure VirtualMachine size
[16] Azure VirtualMachine size
[17] CRISP DM
[18] Azure ml studio
[19] Mlflow
[20] Datastore
[21] Model Registry
[22] Environments
[23] Endpoints
[24] XGboost architecture
[25] Random forest
[26] KNN
[27] InceptionV3
[28] VGG19
[29] CNN
[30] EfficientNetB0
[31] SVM
[32] EfficientNetB3