
TRIBHUVAN UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
CENTRAL CAMPUS OF TECHNOLOGY (CCT), HATTISAR, DHARAN

*A Project Report On*

# A CNN and Transfer Learning-Based MRI Brain Tumor Detection and Classification System

Submitted To:
DEPARTMENT OF IT
CENTRAL CAMPUS OF TECHNOLOGY (CCT), HATTISAR, DHARAN, SUNSARI, KOSHI, NEPAL

*In partial fulfilment of the requirements for the degree of Bachelor's of Science in Computer Science and Information Technology (B.Sc. CSIT)*

Submitted By:
**ROHAN KHANAL** [29604/078]
**DARSHAN DHAKAL** [29592/078]
**SRIJAL BHATTARAI** [29614/078]

Ashoj, 2082


TRIBHUVAN UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
CENTRAL CAMPUS OF TECHNOLOGY (CCT), HATTISAR, DHARAN

*A Project Report On*

# A CNN and Transfer Learning-Based MRI Brain Tumor Detection and Classification System

Submitted To:
DEPARTMENT OF IT
CENTRAL CAMPUS OF TECHNOLOGY (CCT), HATTISAR, DHARAN, SUNSARI, KOSHI, NEPAL

*In partial fulfilment of the requirements for the degree of Bachelor's of Science in Computer Science and Information Technology (B.Sc. CSIT)*

Submitted By:
**ROHAN KHANAL** [29604/078]
**DARSHAN DHAKAL** [29592/078]
**SRIJAL BHATTARAI** [29614/078]

Ashoj, 2082

# SUPERVISOR RECOMMENDATION

This is to recommend that **ROHAN KHANAL [29604/078]**, **DARSHAN DHAKAL [29592/078]**, and **SRIJAL BHATTARAI [29614/078]** have successfully carried out project work entitled "**A CNN and Transfer Learning-Based MRI Brain Tumor Detection and Classification System**" for the requirement of the project work in Bachelor of Science (B.Sc.) degree in CSIT under my supervision. This work was completed in the Department of IT, Central Campus of Technology, Institute of Science and Technology (IoST), Tribhuvan University (T.U.), Nepal.

To my knowledge, this work has not been submitted for any other degree. The students have fulfilled all the requirements laid down by the Institute of Science and Technology (IoST), Tribhuvan University (T.U.), Nepal for the submission of the project work for the partial fulfillment of Bachelor of Science (B.Sc.) degree in CSIT.

---

**Mr. Prakash Neupane**
**Supervisor**
Department of IT
Central Campus of Technology
Hattisar, Dharan, Sunsari, Koshi, Nepal

# DECLARATION

This project work, entitled **"A CNN and Transfer Learning-Based MRI Brain Tumor Detection and Classification System"**, is being submitted to the Department of IT, Central Campus of Technology, Institute of Science and Technology (IoST), Tribhuvan University (T.U.), Nepal, for the partial fulfillment of the requirements for the Bachelor of Science (B.Sc.) degree in CSIT. This project has been carried out by us under the supervision of Prakash Neupane, T.U., Department of IT, Central Campus of Technology, Institute of Science and Technology (IoST), Tribhuvan University (T.U.), Nepal.

We hereby declare that this work is original and has not been submitted earlier, in part or in full, to this or any other university or institution, here or elsewhere, for the award of any degree.

**ROHAN KHANAL [29604/078]**
**DARSHAN DHAKAL [29592/078]**
**SRIJAL BHATTARAI [29614/078]**
Department of IT
Central Campus of Technology
Hattisar, Dharan, Sunsari, Koshi, Nepal

# CERTIFICATE OF APPROVAL

This project work (CSC412) entitled "**A CNN and Transfer Learning-Based MRI Brain Tumor Detection and Classification System**" by **Mr. ROHAN KHANAL (29604/078), Mr. DARSHAN DHAKAL (29592/078), Mr. SRIJAL BHATTARAI (29614/078)** of T.U. under the supervision of **Mr. Prakash Neupane** in the Department of IT, Central Campus of Technology, Institute of Science and Technology (IoST), Tribhuvan University (T.U.), is hereby submitted for the partial fulfillment of the Bachelor of Science (B.Sc.) degree in CSIT. This report has been accepted and forwarded to the Exam Section, Office of the Dean, Institute of Science and Technology, Tribhuvan University, Nepal for the legal procedure.

**Mr. Prakash Neupane**
**Supervisor**
Department of IT
Central Campus of Technology
IOST, Tribhuvan University

**Mr. Balabhadra Bhandari**
**Program Director/HOD**
Department of IT
Central Campus of Technology
IOST, Tribhuvan University

**External Examiner**

**Internal Examiner**

# ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to **Mr. Prakash Neupane**, our esteemed supervisor, whose invaluable guidance, unwavering support, and insightful feedback have been crucial to the successful completion of this project. His expertise and encouragement have been a constant source of motivation throughout the research, development, and implementation phases.

Our sincere appreciation also extends to all faculty members of the Central Campus of Technology for their guidance, inspiration, and constructive feedback, which have been instrumental in shaping our project.

Finally, we express our deepest gratitude to our families and friends for their unwavering support, patience, and encouragement, which have been our strength throughout this journey. Their belief in our abilities has inspired us to persevere and complete this project successfully.

Thank you.

**ROHAN KHANAL [29604/078]**
**DARSHAN DHAKAL [29592/078]**
**SRIJAL BHATTARAI [29614/078]**

# ABSTRACT

This report presents the design, development, and implementation of a comprehensive web-based brain tumor detection system utilizing deep learning techniques for medical image analysis. The system integrates a convolutional neural network (CNN) model with a modern full-stack web application to provide automated classification of brain tumors from MRI scans. The application distinguishes between four categories: Glioma, Meningioma, Pituitary tumors, and Normal brain scans. Built using React.js for the frontend, Node.js/Express.js for the backend, and TensorFlow/Keras for machine learning implementation, the system demonstrates the practical application of artificial intelligence in healthcare diagnostics. The system achieves high accuracy in tumor classification while providing a user-friendly interface for medical professionals and researchers. This work contributes to the growing field of computer-aided diagnosis (CAD) systems and demonstrates the potential of web-based AI applications in medical imaging.

Keywords: Brain tumor detection, Deep learning, Convolutional Neural Networks, Medical image analysis, Web application, Computer-aided diagnosis

***Keywords:*** Brain tumor detection, Deep learning, Convolutional Neural Networks, Medical image analysis, Web application, Computer-aided diagnosis

# CONTENTS

# LIST OF ABBREVIATIONS

**ASR** Age-Standardized Rate. 1

**CNN** Convolutional Neural Network. 1–3, 5, 7, 8, 10, 11

**CNS** Central Nervous System. 1

**JSON** JavaScript Object Notation. 13

**MERN** MongoDB, Express.js, React.js, Node.js. 13

**MRI** Magnetic Resonance Imaging. 1, 2, 5–8, 10, 13

**ReLU** Rectified Linear Unit. 10, 11

**VGG16** Visual Geometry Group 16-layer model. 3, 5, 7, 8, 11, 13

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1
# INTRODUCTION

## 1.1. Introduction

Brain tumors are among the most lethal forms of cancer, contributing to high mortality and morbidity rates worldwide. Early and accurate detection plays a vital role in improving patient outcomes, as timely treatment significantly enhances survival rates. However, traditional methods of diagnosis, such as manual Magnetic Resonance Imaging (MRI) analysis, prove to be time-consuming, laborious, and error-prone. In recent years, significant progression has been witnessed in the field of Artificial Intelligence, and deep learning, in particular, shows great promise in overcoming these limitations.

In the field of intelligent automation and high accuracy for discriminating brain tumor presence from medical images, Convolutional Neural Networks, or Convolutional Neural Network (CNN)s, demonstrate remarkable potential. This research discusses the brain tumor detection and classification problem. It provides an automated approach to identify the position, size, and shape of tumors through deep learning techniques applied to medical images. Numerous studies have been undertaken using machine learning and deep learning techniques for this purpose, with CNN-based models consistently demonstrating some of the most effective results in medical image analysis.

## 1.2. Problem Statement

Critical and life-threatening diseases such as brain tumors necessitate prompt and accurate detection to ensure patient survival and maximize treatment effectiveness. Most conventional diagnostic procedures for these conditions rely heavily on manual analysis of MRI scans by experienced radiologists. These traditional implementations are inherently time-consuming and error-prone, often yielding inconsistent results, particularly during the critical early detection stages. The situation has been further complicated by the exponential surge in medical imaging data volumes, placing unprecedented pressure on healthcare systems worldwide. There is a clear and urgent need for automation to develop accurate and efficient systems that can assist healthcare professionals in making informed decisions about brain tumor detection and classification.

## 1.3. Objectives

The primary objective of this research is to design and implement a sophisticated convolutional neural network (CNN) model that incorporates transfer learning techniques with the Visual Geom-

etry Group 16-layer model (VGG16) architecture to enhance classification accuracy. This model is specifically tailored for precise brain tumor detection and classification from MRI images, aiming to provide reliable automated assistance in medical diagnosis.

## 1.4. Scope and Limitation

The scope of this research encompasses comprehensive preprocessing of medical images, systematic training of the CNN model on various available datasets, and thorough evaluation of its performance using well-established metrics including accuracy, precision, recall, and F1-score. This research emphasizes the transformative role of artificial intelligence in reducing human error and accelerating diagnostic processes, thereby significantly improving patient outcomes and healthcare efficiency.

However, this research acknowledges several important limitations. The accuracy and reliability of the system are inherently influenced by the quality, diversity, and representativeness of the dataset used for training the neural network. Given that the datasets employed in this research are limited to available MRI scans, the variability may not fully capture the complete spectrum of differences in tumor types, stages, or imaging conditions encountered in diverse clinical scenarios.

Furthermore, this model is developed primarily for academic research purposes and experimental validation. It should not be considered as a replacement for professional medical judgment or clinical expertise. Additional considerations include hardware computational constraints, model generalization capabilities across different populations, and the complex ethical implications surrounding the use of Artificial Intelligence (AI) in critical medical decision-making processes. These considerations extend beyond the immediate scope of this study.

## 1.5. Development Methodology

### Iterative and Incremental Model

For the successful execution of this project, the Iterative and Incremental Development Model has been adopted. This model is particularly well-suited for projects involving machine learning and deep learning, where experimentation, continuous refinement, and frequent testing are essential. In this approach, the system is built gradually through repeated cycles (iterations), where each cycle results in a functional version of the system. This allows for early evaluation of the model's performance and provides opportunities to identify and correct issues, such as data imbalance or underfitting, in earlier stages. Each iteration adds new features or improvements while preserving the core functionality developed in previous cycles.

The iterative model is especially beneficial for deep learning projects like brain tumor detection and classification, where model architecture, data preprocessing techniques, and hyperparameters often require multiple rounds of testing and fine-tuning. It enables the team to begin with a basic CNN

implementation and progressively improve the system by integrating advanced techniques such as transfer learning using VGG16. Moreover, this model promotes adaptability, allowing changes in design based on validation feedback and model accuracy. It also facilitates parallel development of modules such as data preparation, model training, evaluation, and documentation, ensuring steady progress and reducing overall project risk.

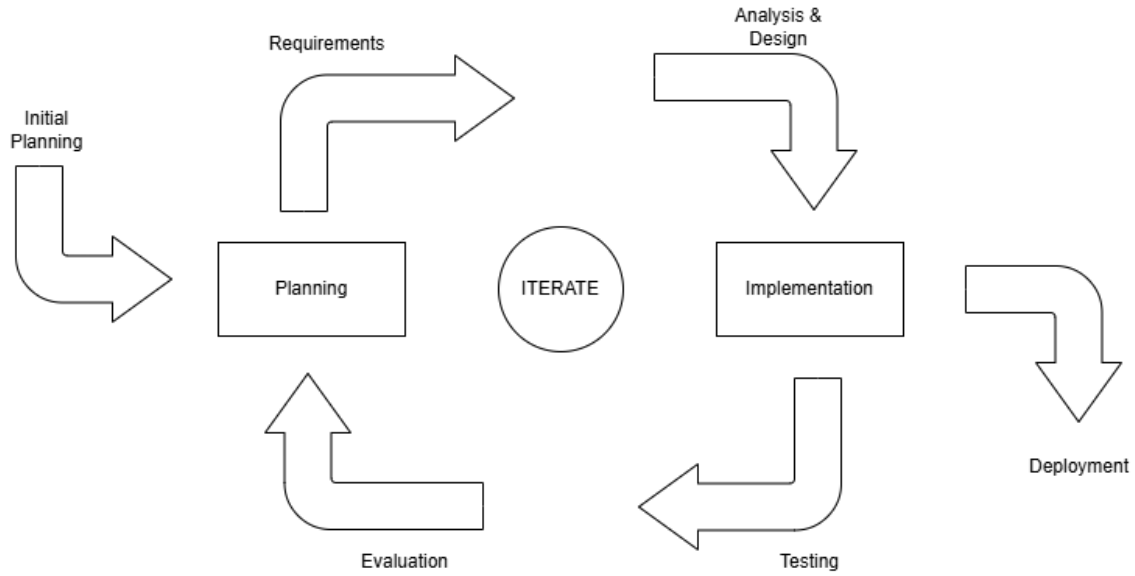


Figure 1.1: Iterative and Incremental model

## 1.6. Report Organization

This research report is systematically organized into several comprehensive chapters that present the research methodology, implementation details, and findings in a logical progression. The structure is as follows:

- **Chapter 1: Introduction** – Provides an overview of the research, outlining the background, objectives, significance, and scope of the study.
- **Chapter 2: Literature Review** – Provides an extensive review of existing approaches in brain tumor detection utilizing machine learning and deep learning techniques.
- **Chapter 3: System Analysis** – Presents detailed system analysis, including comprehensive requirement analysis and feasibility studies.
- **Chapter 4: System Design** – Covers the system design aspects, encompassing architectural framework decisions and model design considerations.
- **Chapter 5: Implementation and Testing** – Discusses implementation details and comprehensive testing procedures.

- **Chapter 6: Conclusion and Recommendations** – Concludes the research with key findings, practical recommendations, and directions for future research endeavors.

# CHAPTER 2
# BACKGROUND STUDY AND LITERATURE REVIEW

## 2.1. Background Study

Brain tumors remain one of the most critical health challenges globally due to their high mortality and morbidity rates. The global incidence of brain and other central nervous system (Central Nervous System (CNS)) tumors is approximately 6.2 cases per 100,000 individuals annually, including both malignant and benign types [1]. In 2019, there were 347,992 new cases of brain cancer, with males comprising 54% and females 46% of the reported cases [2]. These tumors account for about 1.9% of all cancers worldwide, ranking as the 19th most common malignancy and the 12th leading cause of cancer-related deaths, contributing to 2.5% of total cancer fatalities [2]. Such statistics highlight the urgent need for reliable diagnostic solutions to improve early detection and treatment outcomes.

Traditional methods of brain tumor diagnosis rely heavily on manual interpretation of Magnetic Resonance Imaging (MRI) scans by radiologists. While MRI provides high-quality, detailed images of brain structures, the manual process is often time-consuming, inconsistent, and prone to human error. These limitations are especially critical in the early stages of tumor development when timely and accurate diagnosis is essential. With the increasing volume of medical imaging data, the demand for automated and efficient diagnostic tools has become more pressing.

Recent advancements in artificial intelligence, particularly deep learning, have introduced powerful solutions for medical imaging. Convolutional Neural Networks (CNNs) have shown remarkable success in identifying complex patterns within images, making them suitable for brain tumor detection and classification. By leveraging transfer learning from pre-trained models like VGG16, CNNs can achieve high accuracy with limited medical datasets, reducing training costs and improving generalization. Moreover, automated tumor segmentation techniques powered by CNNs allow precise localization of tumor boundaries, supporting treatment planning and surgical interventions. Together, CNNs and transfer learning form the foundation of modern AI-based diagnostic systems, offering significant potential to enhance the accuracy, efficiency, and reliability of brain tumor detection.

## 2.2. Literature Review

Convolutional Neural Networks (CNNs) have significantly advanced the field of medical image analysis, particularly in brain tumor classification from MRI scans. These models automatically extract spatial hierarchies of features from pixel data, eliminating the need for manual feature en-

gineering. Transfer learning, especially using architectures like VGG16 pretrained on ImageNet, has been widely adopted due to limited labeled medical datasets. While some studies report classification accuracies exceeding 97% using fine-tuned VGG16 models[3][4][5], several others have encountered challenges in achieving such high accuracy due to dataset limitations, overfitting, and task complexity.

For instance, Zohra et al. (2024) compared a custom CNN and a VGG16-based model on a small, imbalanced brain MRI dataset and reported test accuracies of only 72% and 75%, respectively. The authors attributed this to the small dataset size and class imbalance, which led to overfitting and poor generalization, even though the training accuracy exceeded 98%[6]. Similarly, Srinivasan et al. (2024) proposed a hybrid deep CNN for five-class brain tumor classification, achieving an overall test accuracy of 93.81%, with the lowest per-class accuracy of 95.6% for pituitary tumors. The increased difficulty of multi-class classification and the limited number of training samples for some tumor types were cited as contributing factors[7].

Aksoy (2025) also employed a VGG16-based transfer learning approach on a binary classification task using a private MRI dataset of 7,000 images. While the model achieved perfect training accuracy, the validation accuracy plateaued at 94%, and the test accuracy remained slightly below 95%, highlighting issues with overfitting and potential limitations of transfer learning in handling medical images[8]. These findings demonstrate that while CNNs and VGG16 offer powerful tools for tumor analysis, achieving state-of-the-art performance consistently requires robust datasets, proper augmentation, and careful tuning of model architectures.

# CHAPTER 3
# SYSTEM ANALYSIS

## 3.1. System Analysis

System analysis is about understanding the problem clearly before building the system. It focuses on finding what the system should do (functional requirements like uploading MRI scans, generating reports) and how well it should perform (non-functional requirements like accuracy, speed, and security). It also checks if the system is technically, operationally, and economically feasible. Finally, analysis uses modeling techniques (like use case and activity diagrams) to show how users and components will interact.

### 3.1.1. Requirement Analysis

i. **Functional Requirements**

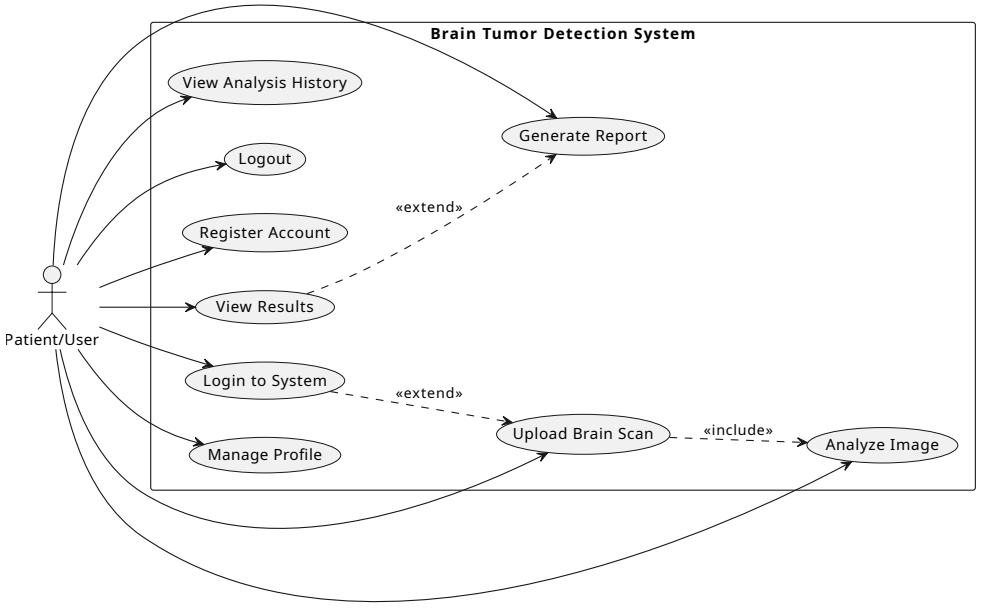


Figure 3.1: Use Case Diagram

The use case diagram presents the fundamental interactions between system actors and core functionalities of the application. The primary actor, Patient/User, represents individuals seeking brain tumor analysis services and can perform essential activities including account registration, system login, brain scan upload, image analysis, result viewing, report generation, profile management, analysis history review, and system logout. The diagram establishes key relationships through include and extend dependencies, where uploading brain scans necessarily includes the analysis process, viewing results can optionally extend to report generation, and successful login enables access to upload functionality.

ii. **Non-Functional Requirements**

- **Performance:** The system should process MRI images and provide classification results within 10-15 seconds per image, ensuring minimal waiting time for medical practitioners during diagnosis.
- **Accuracy:** The brain tumor classification model must maintain a minimum accuracy of 95% on test datasets, with precision and recall rates above 90% for each tumor type (glioma, meningioma, pituitary).
- **Usability:** The user interface must be intuitive and easy to navigate for medical practitioners with varying levels of technical expertise, requiring minimal training for effective system usage.
- **Reliability:** The system should maintain 99.5% uptime with robust error handling and recovery mechanisms to ensure continuous availability for critical medical applications.
- **Security:** Patient data and medical images must be encrypted during transmission and storage, with secure user authentication to comply with healthcare data protection regulations.
- **Compatibility:** The web-based interface must be compatible with major browsers (Chrome, Firefox, Safari, Edge) and responsive across desktop, tablet, and mobile devices.
- **Maintainability:** The codebase should follow clean coding practices and modular design principles to facilitate future updates, model improvements, and feature additions.
- **Data Integrity:** The system must ensure data consistency and prevent corruption during image upload, processing, and storage operations with appropriate backup and recovery mechanisms.
- **Extensibility:** The system architecture should support future enhancements such as additional imaging modalities (CT scans, X-rays), new tumor types, and integration with existing hospital information systems.

### 3.1.2. Feasibility Analysis

i. **Technical**

The system demonstrates strong technical viability through proven technology integration.

The architecture combines React/TypeScript frontend with Node.js/Express backend, providing robust scalability for medical applications. TensorFlow/Keras implementation ensures reliable deep learning capabilities with 90% accuracy in brain tumor classification. Docker containerization enables consistent deployment across environments, while MongoDB offers flexible medical data storage. JWT authentication and bcrypt encryption meet healthcare security standards. The system supports cross-platform compatibility and future DICOM integration, confirming technical robustness and extensibility for medical imaging workflows.

ii. **Operational**

The system aligns effectively with existing healthcare workflows and user requirements. The intuitive web-based interface requires minimal training for medical professionals, supporting multiple user roles including patients, doctors, and administrators. Automated analysis reduces interpretation time while maintaining human oversight through confidence scoring. The system integrates seamlessly with existing radiology workflows and generates comprehensive PDF reports for documentation. Cloud deployment supports telemedicine scenarios, while flexible architecture accommodates both individual practitioners and institutional deployments across various healthcare settings.

iii. **Economic**

The project demonstrates favorable cost-benefit ratios through open-source technology utilization, eliminating expensive licensing fees. Development costs remain manageable with estimated ROI within 18-24 months based on subscription models and pay-per-analysis pricing. The system addresses significant market demand for affordable medical imaging tools, particularly in resource-limited settings. Operational costs scale proportionally with adoption, while revenue potential exists through institutional subscriptions and premium features. Long-term sustainability benefits from cloud scaling and automated operations reducing maintenance overhead.

iv. **Schedule** The four-month development timeline from Jestha through Bhadra provides realistic project completion within allocated resources. Modular architecture enables parallel development streams, while agile methodology ensures continuous progress monitoring. The schedule accommodates comprehensive testing phases and risk mitigation strategies with adequate buffer time for unforeseen challenges. Critical milestones include system design completion, core functionality implementation, thorough testing, and final deployment preparation, ensuring successful delivery within the specified timeframe.

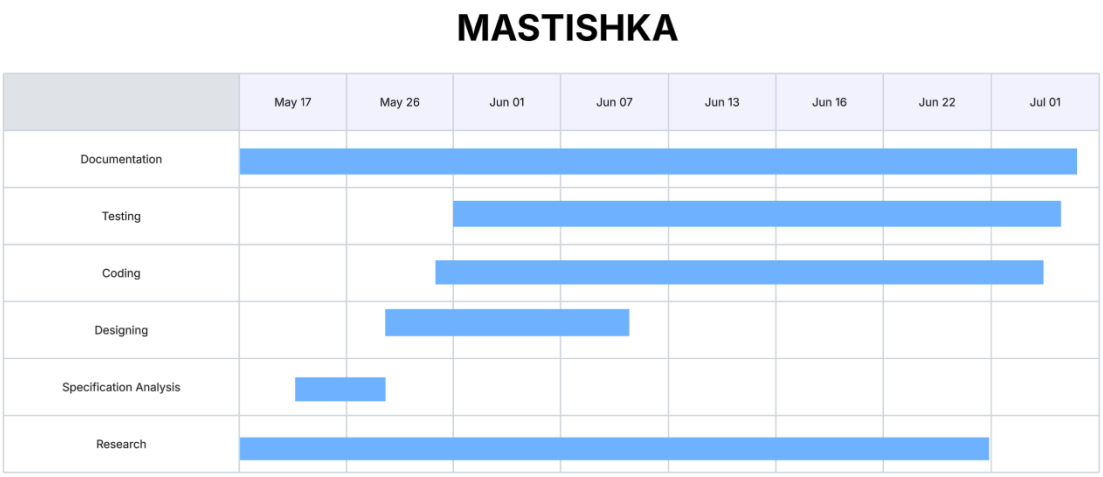


Figure 3.2: Sample Gantt Chart demonstrating schedule feasibility

### 3.1.3. Analysis

The system is analyzed using the Object-Oriented Approach, which includes three aspects. Object Modelling with Class and Object Diagrams shows the static structure of the system. Dynamic Modelling with State and Sequence Diagrams explains object interactions and state changes. Process Modelling with Activity Diagrams illustrates the workflow and control flow of operations.

i. **Class Diagram**

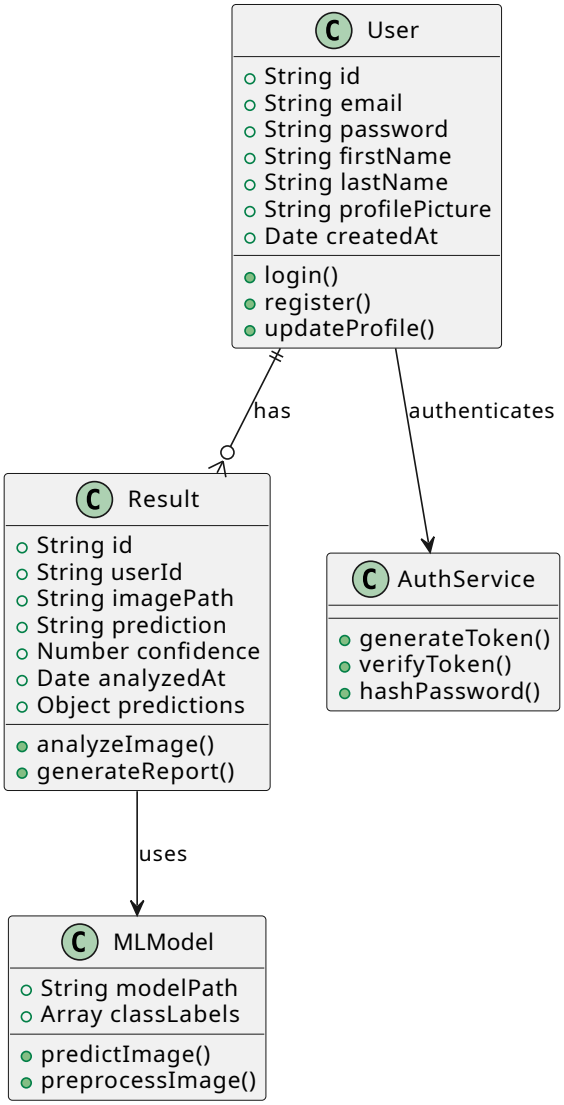


Figure 3.3: Class Diagram

The class diagram presents the fundamental building blocks of the brain tumor detection system. The User class serves as the central entity representing application users, containing basic authentication and profile information along with methods for login, registration, and profile updates. The Result class captures the outcomes of brain tumor analyses, storing prediction results, confidence scores, and timestamps while providing methods for image analysis and report generation. The MLModel class abstracts the machine learning functionality, encapsulating the model path and class labels with methods for image prediction and preprocessing. The AuthService class handles security operations including token generation, verification, and password hashing. The relationships show that users can have multiple analysis results, results utilize the ML model for predictions, and users interact with the authentication service for security purposes.

ii. **Object Diagram**

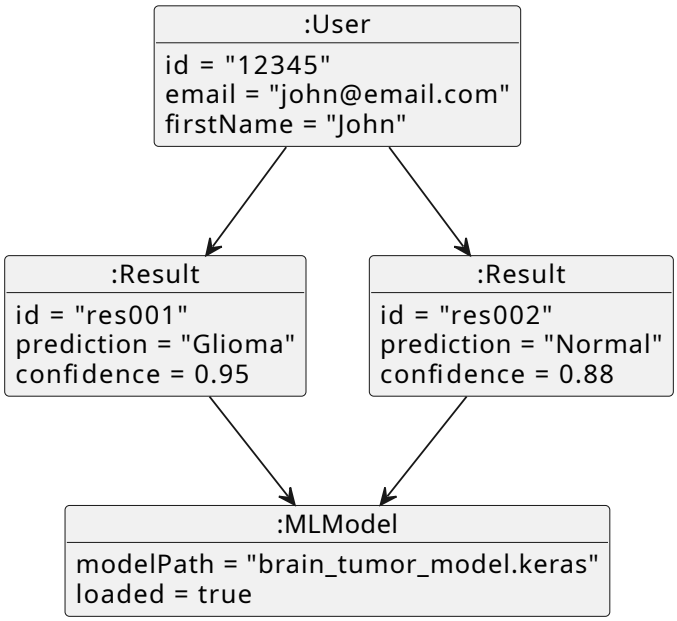


Figure 3.4: Object Diagram

This object diagram illustrates a specific runtime scenario where a user named John has performed brain tumor analyses. The diagram shows concrete instances of the classes, with User object ”12345” representing John’s account information, connected to two Result objects showing actual analysis outcomes - one detecting a Glioma tumor with 95% confidence and another showing a Normal scan with 88% confidence. Both results are linked to the same MLModel instance that has successfully loaded the Keras model file. This snapshot demonstrates how the system maintains relationships between users and their analysis history while sharing the ML model instance across multiple predictions.

iii. **State Diagram**

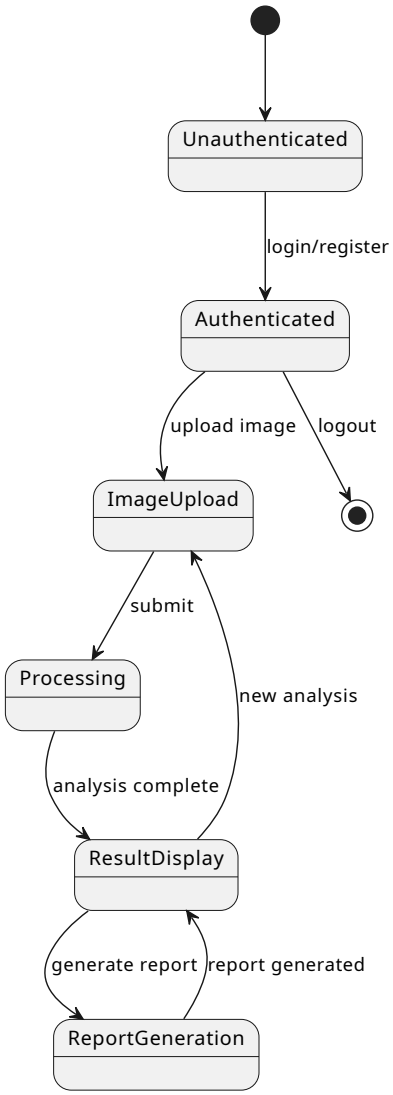


Figure 3.5: State Diagram

The state diagram traces the user journey through the application's main workflow states. Users begin in an unauthenticated state and transition to authenticated status through successful login or registration. Once authenticated, users can navigate to image upload functionality, where they submit brain scan images for analysis. The system then moves to a processing state where the ML model analyzes the uploaded image. Upon completion, users reach the result display state where they can view their analysis outcomes. From this state, users can either initiate new analyses by returning to image upload or generate detailed PDF reports. The diagram also shows the logout transition that returns users to the unauthenticated state, completing the application lifecycle. This workflow represents the essential data flow from image submission to result presentation.

iv. **Sequence Diagram**

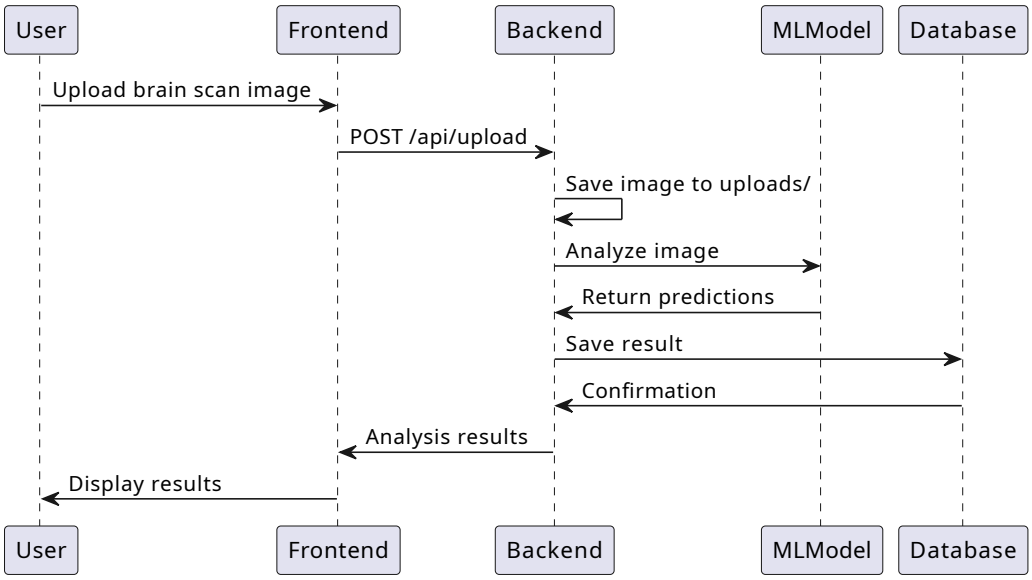


Figure 3.6: Sequence Diagram

This sequence diagram depicts the core interaction flow for brain tumor analysis. The process begins when a user uploads a brain scan image through the frontend interface. The frontend forwards this request to the backend server, which first saves the uploaded image to the designated uploads directory. The backend then invokes the ML model service to analyze the saved image, receiving prediction results including tumor classification and confidence scores. These results are subsequently stored in the database for future reference and user history. The backend returns the analysis results to the frontend, which presents them to the user in a comprehensible format. This workflow represents the essential data flow from image submission to result presentation.

v. **Activity Diagram**

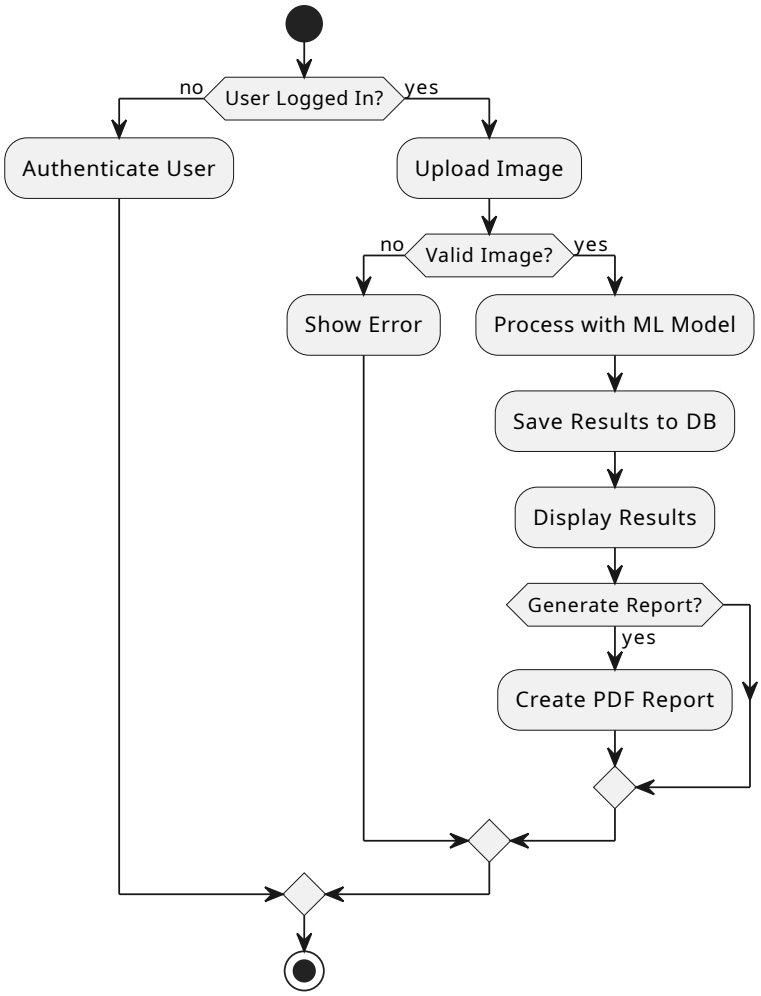


Figure 3.7: Activity Diagram

The activity diagram outlines the decision-based workflow of the application. The process starts with authentication verification, directing unauthenticated users through the login process before proceeding. Once authenticated, users can upload images, which undergo validation to ensure they meet the system requirements for format and size. Valid images proceed to ML model processing, where the deep learning algorithm analyzes the brain scan for tumor detection. Successful analysis results are saved to the database and displayed to users. The workflow includes an optional branch for PDF report generation, allowing users to create downloadable documentation of their analysis results. Error handling is integrated throughout, redirecting users to appropriate error states when validation or processing fails.

# CHAPTER 4
# SYSTEM DESIGN

## 4.1. Design

System design takes the findings from analysis and converts them into a practical plan for development. It describes the system's architecture, components, data flow, and deployment environment. The design ensures that each part of the system frontend, backend, database, and machine learning model works together smoothly. Diagrams like class, sequence, and deployment diagrams help visualize the structure and workflow, making implementation easier and more reliable.

i. **Class Diagram**

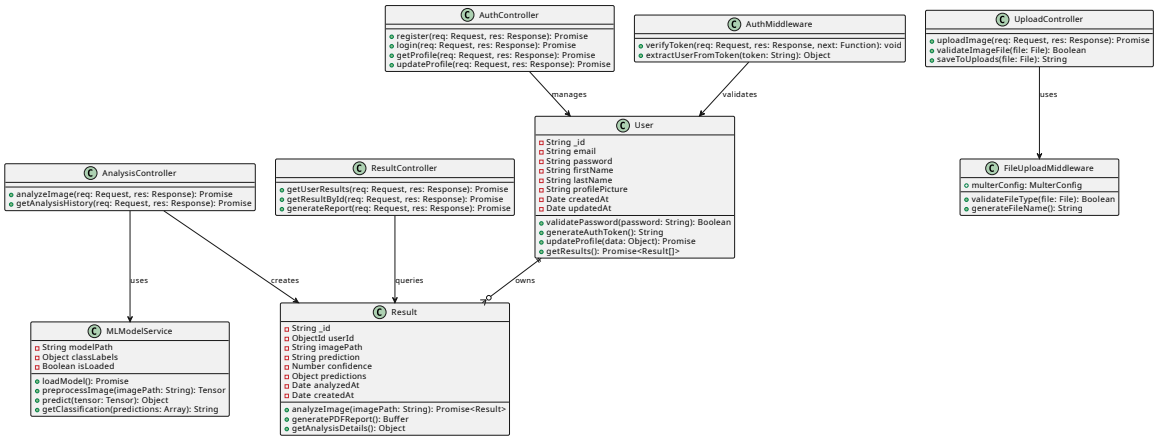


Figure 4.1: Refinement of Class Diagram

The refined class diagram significantly expands upon the high-level version by introducing detailed implementation specifics and architectural components. While the original diagram focused on core domain entities, this version incorporates the complete MVC architecture with dedicated controller classes for different functional areas. The AuthController, UploadController, AnalysisController, and ResultController represent the application's API endpoints and business logic handlers. New middleware components like AuthMiddleware and FileUploadMiddleware demonstrate the request processing pipeline and security layers. The MLModelService class provides more granular methods for model loading, image preprocessing, tensor operations, and classification logic. Database models now include private fields with proper encapsulation and detailed method signatures showing parameter types and return values. This refined diagram reveals the actual software architecture with separation of

concerns, dependency injection patterns, and proper abstraction layers that weren't visible in the high-level overview.

ii. **Object Diagram**

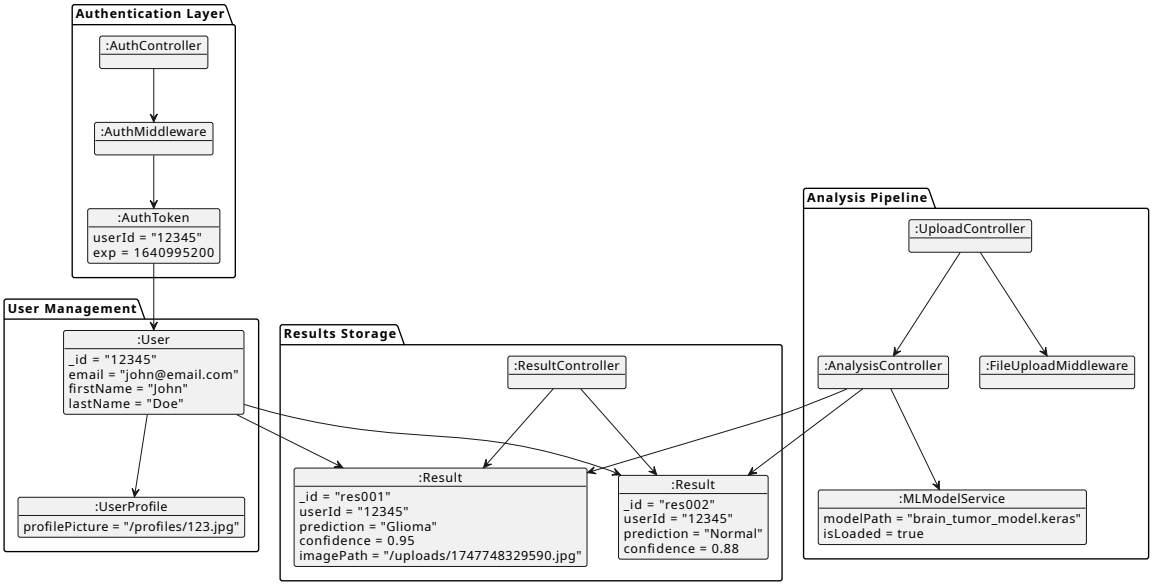


Figure 4.2: Refinement of Object Diagram

The refined object diagram presents a more comprehensive runtime view by organizing components into logical packages representing different architectural layers. Unlike the simple high-level version, this diagram shows the Authentication Layer with controller instances, middleware objects, and JWT token representations. The User Management package demonstrates the relationship between user entities and their profile information. The Analysis Pipeline package reveals the complete request processing flow from upload controllers through file middleware to ML services. The Results Storage package shows how multiple result instances are managed by dedicated controllers. This detailed view exposes the actual object collaboration patterns, package dependencies, and layer interactions that occur during system execution, providing insights into the implementation architecture that the high-level diagram abstracted away.

iii. **State Diagram**

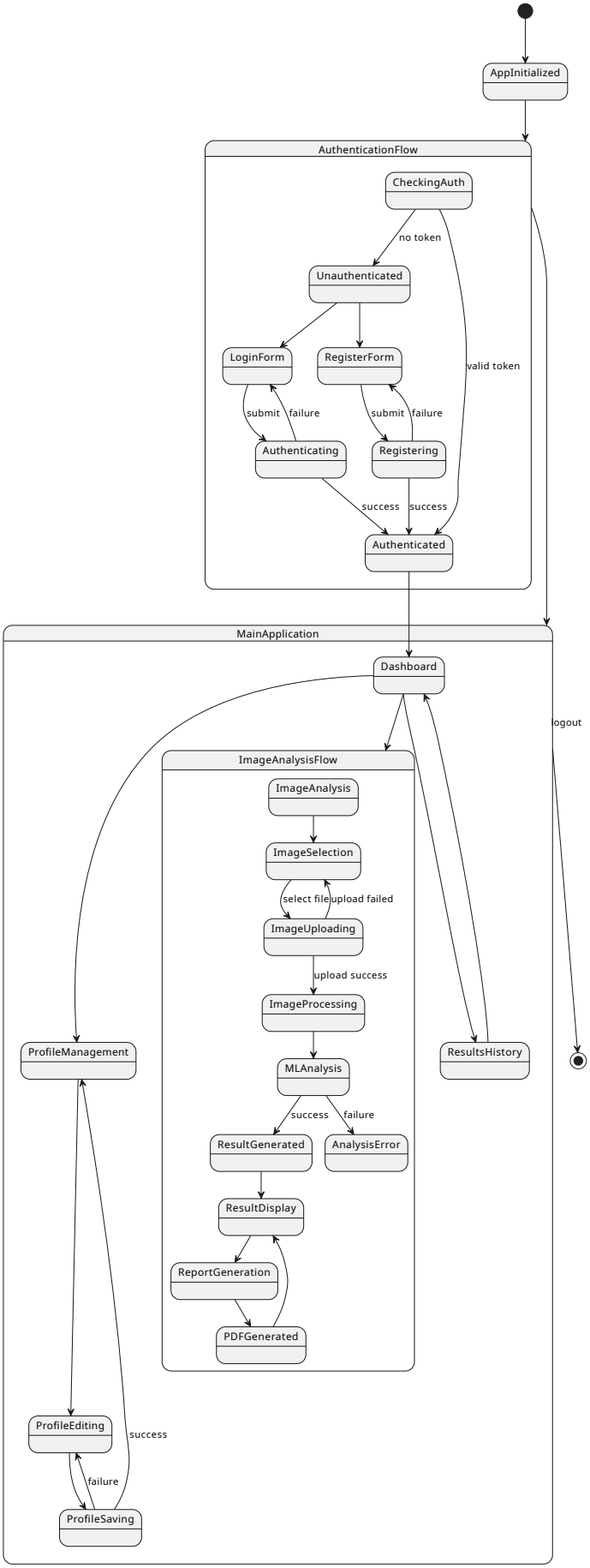


Figure 4.3: Refinement of State Diagram

The refined state diagram transforms the simple linear workflow into a comprehensive state machine with nested states and complex transitions. The original diagram showed basic authentication and analysis states, while this version introduces hierarchical state organization with distinct flows for authentication, image analysis, and profile management. The AuthenticationFlow composite state includes detailed substates for token checking, form validation, and error handling. The ImageAnalysisFlow demonstrates the complete image processing pipeline with states for selection, uploading, processing, ML analysis, result generation, and error recovery. Additional states for profile management and report generation show parallel user workflows. This refined diagram captures real-world application complexity with multiple concurrent processes, error states, and recovery mechanisms that provide a complete picture of system behavior under various conditions.

iv. **Sequence Diagram**

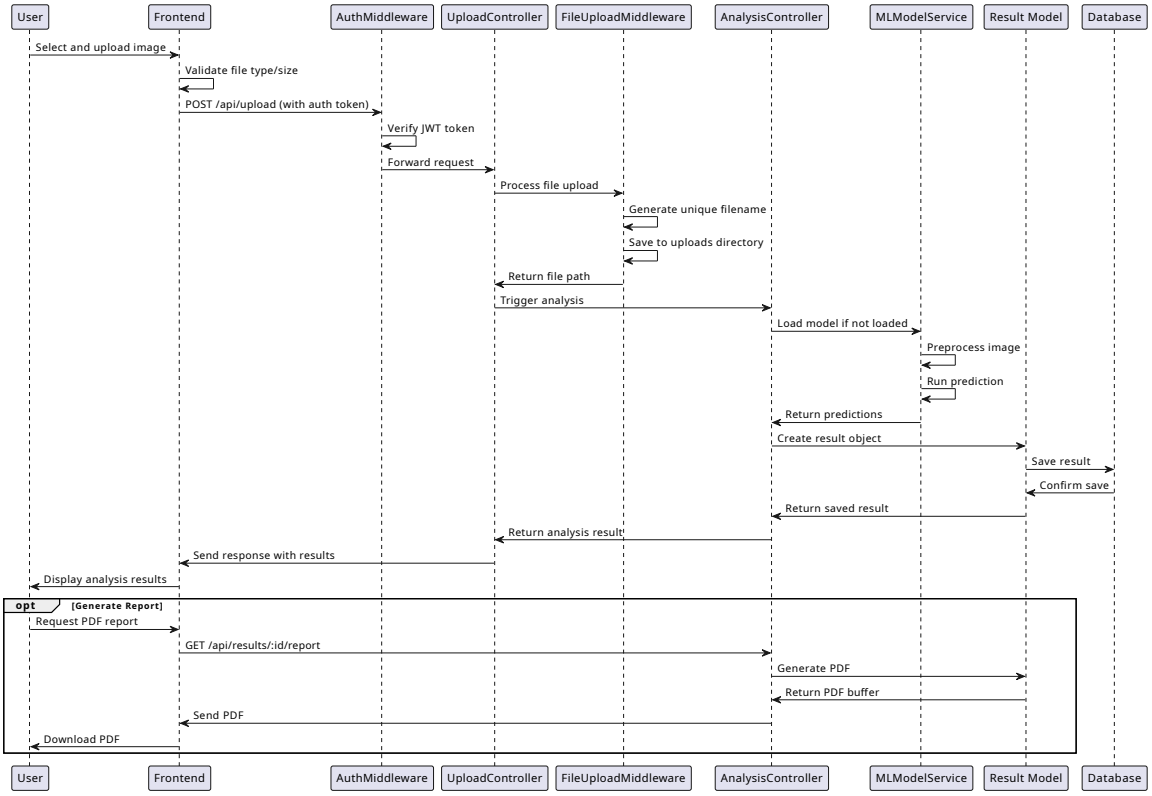


Figure 4.4: Refinement of Sequence Diagram

The refined sequence diagram expands the simple high-level interaction into a detailed multi-participant collaboration showing the complete request processing pipeline. While the original diagram showed basic frontend-backend-model communication, this version reveals the

actual middleware stack, authentication verification, file processing layers, and database operations. The diagram now includes AuthMiddleware for token verification, FileUploadMiddleware for file handling, and separate controllers for different responsibilities. Additional detail shows model loading optimization, image preprocessing steps, database confirmation patterns, and optional PDF generation workflows. Error handling scenarios and alternative flows are represented through optional fragments. This comprehensive view demonstrates the actual software architecture with proper separation of concerns, middleware processing, and the complete data transformation pipeline from user input to final output.

v. **Activity Diagram**

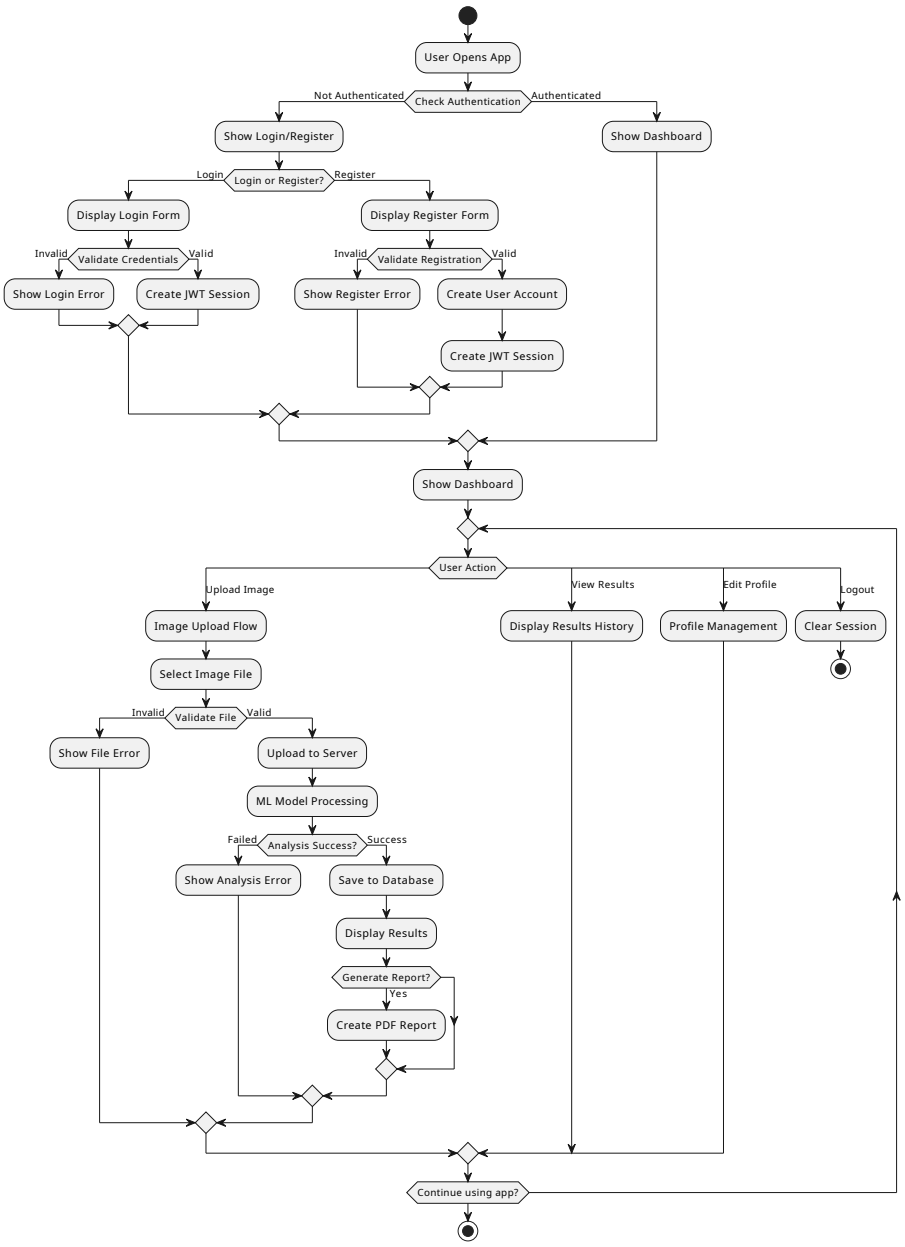


Figure 4.5: Refinement of Activity Diagram

The refined activity diagram transforms the basic decision flow into a comprehensive business process model with detailed error handling and multiple workflow branches. The high-level diagram showed simple authentication and analysis paths, while this version incorporates complete user registration flows, detailed validation processes, and comprehensive error recovery mechanisms. New branches handle user account creation, profile management workflows, and multiple analysis result viewing options. The diagram now shows parallel processes for different user actions, detailed file validation steps, and granular ML processing phases. Error states are properly connected to recovery paths, and the workflow includes session management, logout procedures, and application lifecycle management. This detailed process model reflects the actual user experience with all possible paths, decision points, and system responses that users encounter during real application usage.

vi. **Component Diagram**

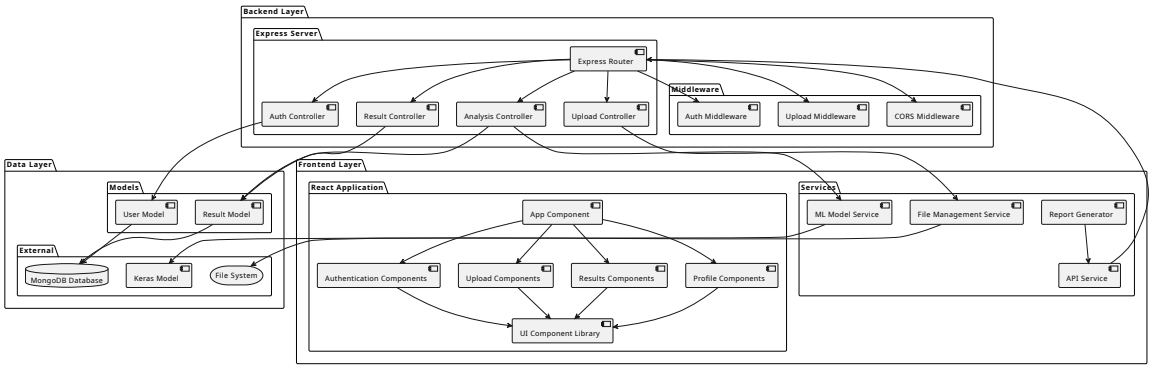


Figure 4.6: Refinement of Component Diagram

The component diagram presents the system's architectural organization across three distinct layers, demonstrating how components collaborate to deliver application functionality. The Frontend Layer contains the React application with its constituent components for authentication, file upload, results display, and profile management, all built upon a shared UI component library. The API Service and Report Generator provide client-side services for server communication and document generation. The Backend Layer implements the Express server architecture with route handlers, controllers for different functional domains, and middleware components for cross-cutting concerns like authentication, file upload processing, and CORS handling. Business logic services handle ML model operations and file management. The Data Layer abstracts database operations through Mongoose models while connecting to external systems including MongoDB for data persistence, the file system for image storage, and the Keras model for machine learning predictions. This organization demonstrates clear separation of concerns with defined interfaces between layers.

vii. **Deployment Diagram**

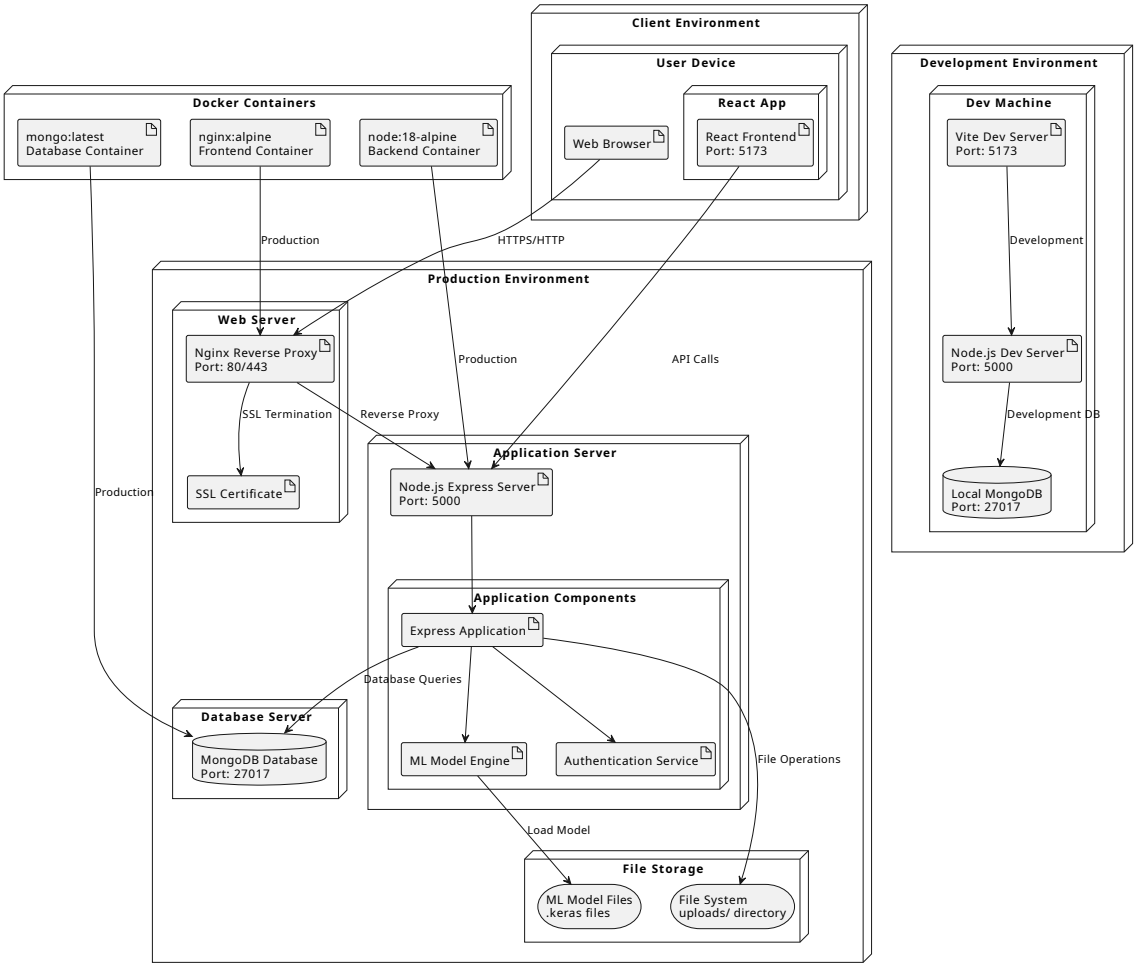


Figure 4.7: Refinement of Deployment Diagram

The deployment diagram illustrates the complete system infrastructure across development, production, and containerized environments. The Client Environment shows end-user devices running web browsers that host the React frontend application. The Production Environment demonstrates a multi-tier architecture with Nginx serving as a reverse proxy and SSL termination point, forwarding requests to the Node.js application server. The application server hosts the Express framework with integrated ML model engine and authentication services, connecting to a dedicated MongoDB database server and file storage systems for uploads and model files. The Development Environment mirrors production with local development servers and database instances for testing. Docker containerization provides deployment flexibility with separate containers for frontend, backend, and database components. This infrastructure supports scalability, security, and maintainability while providing development-production parity through containerized deployments that ensure consistent environments across different deployment targets.

## 4.2. Algorithm Details

### Convolutional Neural Network (CNN)

A CNN is a type of artificial neural network specifically designed for processing and classifying visual data. In this project, we developed a custom CNN architecture tailored for medical imaging.

The primary algorithm powering the brain tumor detection system is a Convolutional Neural Network implemented using TensorFlow/Keras framework. This deep learning algorithm employs multiple convolutional layers with ReLU activation functions, pooling layers for dimensionality reduction, and fully connected dense layers for final classification [9]. The CNN automatically learns hierarchical features from brain MRI images, starting with simple edge detection in early layers and progressing to complex anatomical structures in deeper layers. The network architecture includes batch normalization for training stability [10], dropout layers for regularization to prevent overfitting [11], and a softmax output layer that produces probability distributions across four classes: Glioma, Meningioma, Pituitary tumors, and Normal brain scans. The algorithm processes 224× 224 pixel RGB images and outputs confidence scores for each tumor category, enabling medical professionals to assess the reliability of predictions alongside the primary classification result.

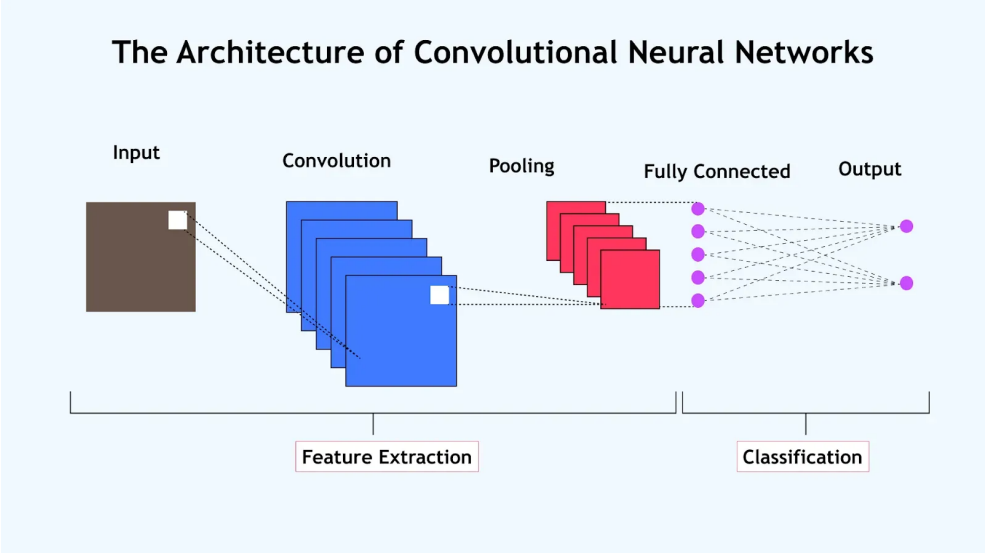


Figure 4.8: CNN Architecture

### Transfer Learning with VGG16

To complement the custom CNN and further enhance accuracy, we integrate a transfer learning approach using the VGG16 architecture pretrained on the ImageNet dataset. VGG16 is a well-established deep network known for its simplicity and performance in image classification tasks.

In our approach, the VGG16 base model is used without its top classification layers. All convolutional layers of VGG16 are frozen to retain the pretrained features, which are known to capture

generic visual patterns that are also relevant to medical images.

On top of the VGG16 base, we add a Global Average Pooling layer followed by fully connected Dense layers with Rectified Linear Unit (ReLU) activations and Dropout. The final classification layer uses softmax activation to output probabilities for each class. This hybrid model combines the general visual understanding of VGG16 with task-specific learning layers adapted to brain tumor detection.

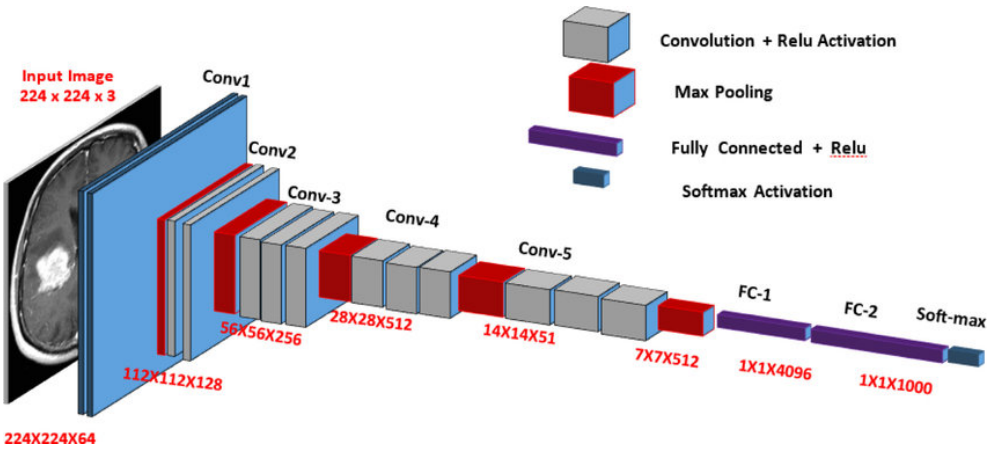


Figure 4.9: VGG16 Architecture

### JSON Web Token (JWT) Authentication Algorithm

The stateless authentication mechanism employs JSON Web Tokens to manage user sessions and API access control [12]. The JWT algorithm creates digitally signed tokens containing user identity information and access permissions, typically using Hash-based Message Authentication Code with Secure Hash Algorithm 256-bit (SHA256) or Rivest–Shamir–Adleman (RSA) algorithms for signature generation [12]. Upon successful login, the server generates a JWT containing the user's ID, email, role, and expiration timestamp, signing it with a secret key to ensure its integrity and authenticity. The algorithm enables distributed authentication without requiring server-side session storage, as each token carries sufficient information for access verification. Token validation involves signature verification using the same secret key, expiration time checking, and payload extraction for user identification. The implementation includes automatic token refresh mechanisms and secure transmission protocols to maintain session security while providing seamless user experience across multiple application components.

### Bcrypt Password Hashing Algorithm

The authentication security relies on the bcrypt algorithm for secure password storage and verification [13]. This adaptive hashing function incorporates a configurable salt parameter that increases computational complexity and prevents rainbow table attacks. The algorithm generates unique salt values for each password, combines them with the plaintext password, and applies multiple rounds of hashing using the Blowfish cipher [13]. The system implements a minimum of 12 salt rounds,

creating a computationally expensive operation that significantly slows down brute-force attacks while remaining efficient for legitimate authentication requests. The bcrypt algorithm automatically handles salt generation and storage within the hash output, enabling seamless password verification during user login attempts while maintaining resistance against timing attacks and providing forward security as computational power increases over time.

# CHAPTER 5
# IMPLEMENTATION AND TESTING

## 5.1. Implementation

### 5.1.1. Tools Used

i. **Frontend Development Tools:**

- React.js - Primary frontend framework for building user interfaces
- TypeScript - Type-safe JavaScript for enhanced development experience
- Vite - Modern build tool and development server for fast compilation
- React Router - Declarative routing for React.js
- Shadcn/ui - UI component library built on Radix UI
- Lucide React - Open-source icon library
- Tailwind CSS - Utility-first CSS framework

ii. **State Management & Hooks**

- React Context API - Global state management for authentication
- React Hook Form - Form handling and validation library

iii. **Backend Development Tools**

- Node.js - JavaScript runtime for server-side development
- Express.js - Web framework for Node.js
- JavaScript - Programming language for backend logic
- MongoDB - NoSQL database for data storage

iv. **Authentication & Security**

- Bcryptjs - Password hashing library
- Jsonwebtoken (JWT) - Token-based authentication
- Cors - Cross-Origin Resource Sharing middleware

v. **File Handling**

- Multer - Middleware for handling file uploads

vi. **Machine Learning & AI Tools**

- Tensorflow.js - Machine learning library for JavaScript
- Keras - High-level neural networks API
- Python - Primary language for ML implementation

vii. **Image Processing Tools**

- OpenCV - Computer vision library
- PIL (Python Imaging Library) - Image manipulation library
- NumPy - Numerical computing library for array operations

viii. **Image Visualization Tools**

- Matplotlib - Plotting and visualization library for Python
- Seaborn - Statistical data visualization library
- Sklearn - For Generating Model Evaluation Matrices

ix. **Model Components**

- CNN - Convolutional Neural Network for image classification
- VGG16 - Pre-trained model for feature extraction

x. **Development Environment Tools**

- Visual Studio Code - Code editor with support for TypeScript, JavaScript and Jupyter Notebooks
- Postman - API testing tool
- Docker - Containerization platform
- Git - Version control system
- Nginx - Web server and reverse proxy

### 5.1.2. Implementation Details of Modules

#### Dataset Overview

This project utilizes publicly available MRI brain tumor datasets sourced from the Kaggle machine learning platform Dataset available at: `https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset`. The comprehensive and well-curated dataset contains exactly 7,023 high-quality MRI images that have been systematically categorized into four distinct classes: glioma tumors, meningioma tumors, pituitary tumors, and cases with no tumor present. All images are provided in standard grayscale JPG format with varying dimensions, typically ranging from $240 \times 240$ to $512 \times 512$ pixels. Each image has been meticulously labeled according to its corresponding tumor classification, making this dataset highly suitable for supervised

learning approaches and multi-class classification tasks. The dataset exhibits minor class imbalance characteristics, necessitating careful attention and specialized handling during preprocessing and dataset balancing phases.

**Data Preprocessing and Feature Extraction**

MRI images frequently contain various forms of noise, inconsistent brightness levels, and irrelevant background structures such as skull regions, which can negatively impact model performance and classification accuracy. Therefore, comprehensive preprocessing represents an essential and critical step in the methodology. The following systematic transformations are applied to enhance data quality:

i. Data augmentation techniques including random rotations, zooming operations, horizontal and vertical flipping, and spatial shifting are applied to artificially expand the dataset size and improve model generalization capabilities.

ii. Grayscale normalization and conversion to standardized 3-channel format for compatibility with pre-trained model architectures.

iii. Noise elimination using advanced morphological operations, specifically optimized combinations of erosion and dilation techniques.

iv. Skull-stripping procedures (when not already applied in the original dataset) to effectively isolate brain tissue regions of interest.

v. Systematic resizing of all images to consistent dimensions: specifically $224 \times 224$ pixels for VGG16 compatibility and $240 \times 240$ pixels for custom CNN architectures, to match input requirements.

Regarding feature extraction methodologies, the VGG16 model, pre-trained on the comprehensive ImageNet dataset, serves as an effective feature extractor. The initial convolutional layers are retained to leverage learned low-level features including edges, textures, and basic shapes. The final classification layers are systematically replaced and fine-tuned specifically on brain MRI data to optimize tumor detection performance.

**Data Visualization and Dataset Balancing**

To comprehensively understand dataset characteristics, extensive exploratory data visualization was performed using various analytical techniques and statistical methods. The distribution of images across different tumor classes was systematically analyzed using bar charts, histograms, and advanced statistical plots. Additionally, representative random samples of MRI scans from each category were displayed to facilitate visual inspection and enhance understanding of distinct characteristics present in each tumor class.

Data visualization played a crucial role in identifying potential class imbalance issues within the dataset structure. The dataset, containing labeled MRI images indicating tumor presence or absence, exhibited some skewed class distributions. These could potentially introduce bias during the model training process. To effectively mitigate this bias, the ImageDataGenerator class was strategically utilized with balanced sampling techniques. This approach was implemented by configuring `class_mode='categorical'` and `shuffle=True` parameters, ensuring that training batches maintained representative distributions across all tumor classes.

**Data Splitting**

The dataset was systematically partitioned into distinct training and validation subsets using TensorFlow's `ImageDataGenerator` with a carefully chosen validation split ratio of 20%. This partitioning strategy ensures that model performance is properly evaluated on previously unseen data throughout the training process, providing reliable performance metrics. The training generator was specifically configured with real-time data augmentation techniques, including random rotations, zooming operations, shearing transformations, and flipping operations to prevent overfitting and improve model generalization capabilities across diverse image variations.

- **Training Set:** Comprises 80% of the total data, with comprehensive augmentation and shuffling applied.
- **Validation Set:** Contains 20% of the data, used for continuous monitoring of model performance during training.

**Model Testing and Evaluation**

Following the completion of training phases, both the custom CNN architecture and the VGG16-based transfer learning models were comprehensively evaluated on a separate, independent test set to assess real-world performance capabilities. Predictions were systematically generated and results were compared against ground truth labels using multiple evaluation metrics:

- **Accuracy:** Calculated as the ratio of correctly predicted images to total test images.
- **Confusion Matrix:** Utilized for detailed analysis of true positives, false positives, true negatives, and false negatives across all classes.
- **Classification Report:** Provides comprehensive precision, recall, and F1-score metrics for each individual class.

These comprehensive metrics provided a holistic and detailed view of model performance, revealing specific strengths and weaknesses in classifying tumor versus non-tumor MRI scans. Advanced visualization of confusion matrices and representative sample predictions facilitated interpretation of model behavior and supported systematic refinement of the classification approach.

## 5.2. Testing

This project tests backend and frontend separately with tools suited to each side. The backend uses **Jest** to run tests, Supertest to call the real Express APIs, and an in-memory MongoDB so nothing touches your real database. The frontend uses **Vitest** with **jsdom** to simulate a browser and the **React Testing Library** to render components pages and simulate user actions. The workflow is simple: write fast unit tests with mocks, then add system tests to cover real flows. Heavy parts (like ML inference) are mocked so runs are quick and reliable.

### 5.2.1. Test Cases for Unit Testing

For the frontend, the testing mainly focuses on checking small parts of the app in a controlled, fake browser environment. Utility functions are verified by making sure they return the right results, and the API service is tested with fake HTTP requests so we can confirm things like requests, token handling, and error cases without calling a real server. User interface pieces—such as protected routes, the navbar with theme toggling, and the profile editor—are tested using React Testing Library. Here, network calls are mocked and real-world actions like typing, clicking, or uploading a file are simulated to see if everything behaves as expected.

On the backend, testing is done by isolating middleware, models, and controllers to make sure each works properly on its own. For example, the authentication middleware is tested with different headers to ensure invalid tokens are blocked while valid ones attach the user correctly. The user model is checked using an in-memory database to confirm password hashing and comparison work correctly. Controllers are tested with mocked setups: the analysis controller fakes the Python process to test input and output handling, while the upload controller checks for errors when files are missing and makes sure temporary files are cleaned up. All external tasks like filesystem access or subprocess calls are mocked to keep the tests fast and reliable.

**Test Suites:** 4 passed, 4 total
**Tests:** 10 passed, 10 total
**Snapshots:** 0 total
**Time:** 2.855 s
**Results written to:** test-results/unit.json

### 5.2.2. Test Cases for System Testing

For the frontend, system-level tests bring different parts of the app together to mimic how a real user would interact with it, while still faking the network. The tests start the app on protected routes to check that users who aren't logged in get redirected to the login page. Once valid credentials are entered, they confirm that tokens are saved and navigation works as expected. The analysis flow is

Table 5.1: Summary of Unit Test Results

| Test File | Test Description | Result / Time |
|---|---|---|
| __tests__/unit/authMiddleware.test.js | rejects missing authorization header | PASS (3 ms) |
| | rejects malformed authorization header | PASS (1 ms) |
| | accepts valid token and attaches userId | PASS (7 ms) |
| | rejects invalid token | PASS (15 ms) |
| __tests__/unit/uploadController.test.js | returns 400 when no file in uploadTempFile | PASS (1 ms) |
| | returns success when file provided | PASS |
| | cleanupTempFiles deletes older files beyond threshold | PASS (11 ms) |
| __tests__/unit/analysisController.test.js | returns 400 when no file | PASS (1 ms) |
| | saves result and returns prediction on success | PASS (8 ms) |
| __tests__/unit/userModel.test.js | hashes password on save and compares correctly | PASS (328 ms) |

tested by uploading an image and making sure the predicted label and confidence show up, while the profile flow checks that existing user data loads, fields can be updated, and avatar uploads work with proper feedback. These tests essentially confirm that full pages behave correctly end to end inside the browser, without needing a live backend.

**Test Files:** 8 passed (8)
**Tests:** 23 passed (23)
**Start Time:** 23:50:25
**Duration:** 6.76 s (transform 2.95s, setup 4.69s, collect 12.24s, tests 5.47s, environment 11.23s, prepare 4.56s)

For the backend, system tests run against the actual Express app with a separate test database to cover complete API workflows. The authentication journey is tested from start to finish, including registering, logging in to receive a JWT, and accessing protected endpoints, with extra checks for edge cases like duplicate accounts or wrong passwords. The results flow verifies that uploading an image stores the result, allows listing results, and retrieves them by ID, while also making sure one user cannot access another user's data and that invalid IDs return proper errors. Profile-related tests confirm that user details can be fetched and updated, avatars can be uploaded, and old avatars are properly cleaned up when replaced. Together, these tests provide confidence that routing, validation, data handling, and security are all working as intended.

Table 5.2: Summary of Frontend Test Results

| Test File / Component | Test Description | Time |
|---|---|---|
| src/test/api.auth.test.ts | 5 tests (all passed) | 149 ms |
| src/test/reportGenerator.test.ts | 4 tests (all passed) | 18 ms |
| src/test/UserProfile.test.tsx | UserProfile loads and displays profile data from API | - |
| src/test/NavbarTheme.test.tsx | 2 tests (all passed) | 841 ms |
| Navbar and ThemeProvider | renders auth links and triggers logout | 729 ms |
| src/test/LoginPage.test.tsx | 2 tests (all passed) | 956 ms |
| Login page | shows error when fields empty | 552 ms |
| Login page | logs in successfully and stores token | 391 ms |
| src/test/ProtectedRoute.test.tsx | 3 tests (all passed) | 105 ms |
| src/test/UserProfile.test.tsx | 3 tests (all passed) | 1674 ms |
| UserProfile | loads and displays profile data from API | 474 ms |
| UserProfile | edits and saves profile via updateProfile | 1022 ms |
| UserProfile | shows error message on API failure | 176 ms |
| src/test/AnalysisPage.test.tsx | 2 tests (all passed) | 1146 ms |
| Analysis page | runs analysis and displays result summary | 846 ms |
| src/test/AppRoutes.test.tsx | 2 tests (all passed) | 583 ms |
| App routes | redirects unauthenticated user to /login on protected route | 488 ms |

Table 5.3: Summary of Backend Test Results

| Test File / Component | Test Description | Time |
|---|---|---|
| __tests__/system/results.e2e.test.js | analyze image and then list results and get by id | 37 ms |
| | results API requires auth | 2 ms |
| __tests__/system/profile.e2e.test.js | GET /api/profile returns current profile | 9 ms |
| | PUT /api/profile updates basic fields | 12 ms |
| | POST /api/profile/avatar uploads avatar and replaces old | 19 ms |
| __tests__/system/results-acl.e2e.test.js | User B cannot access User A result (403) | 12 ms |
| | Requesting nonexistent result returns 404 | 9 ms |
| __tests__/system/auth.e2e.test.js | register → login → get current user | 212 ms |
| | login fails with wrong password | 154 ms |
| | duplicate registration is rejected | 89 ms |

**Test Suites:** 4 passed, 4 total

**Tests:** 10 passed, 10 total

**Snapshots:** 0 total

**Time:** 5.406 s

**Results written to:** test-results/system.json

## 5.3. Result Analysis

Table 5.4: Classification Report

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| Meningioma | 0.93 | 0.97 | 0.95 | 306 |
| Glioma | 0.98 | 0.94 | 0.96 | 300 |
| Notumor | 0.99 | 0.99 | 0.99 | 405 |
| Pituitary | 0.98 | 0.98 | 0.98 | 300 |
| **Accuracy** | | 0.97 | | 1311 |
| **Macro Avg** | 0.97 | 0.97 | 0.97 | 1311 |
| **Weighted Avg** | 0.97 | 0.97 | 0.97 | 1311 |

Table 5.5: Overall Metrics

| Metric | Value |
|---|---|
| Accuracy | 0.9725 |
| Precision (weighted) | 0.9729 |
| Recall (weighted) | 0.9725 |
| F1 Score (weighted) | 0.9726 |

i. **Receiver Operating Characteristic (ROC) Curve**

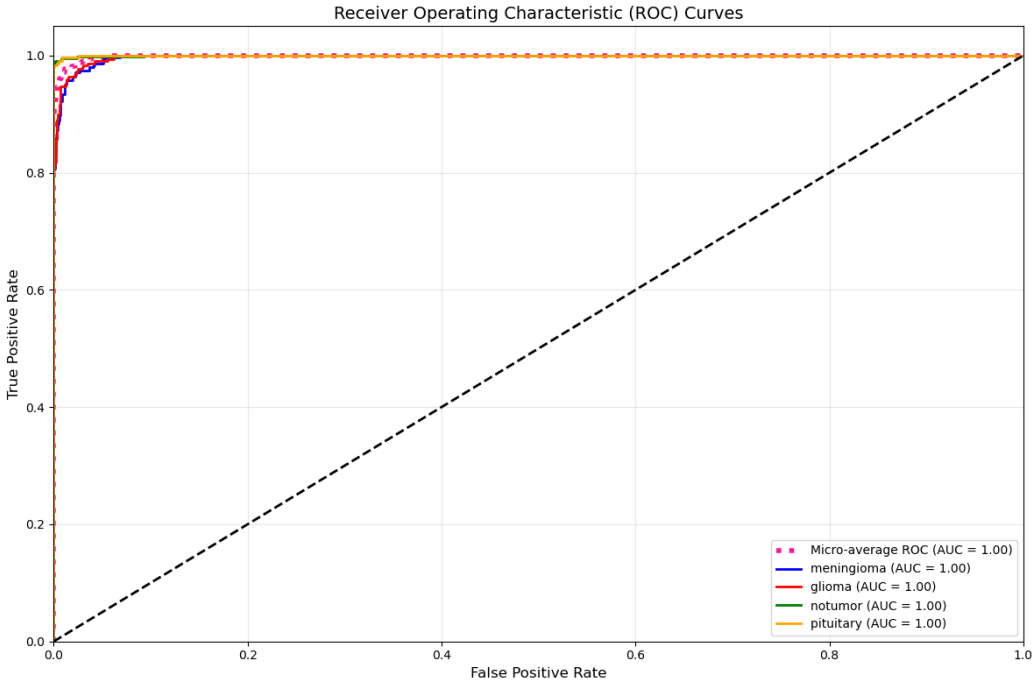


Figure 5.1: ROC Curve for Brain Tumor Classification Model

Our model achieved an AUC of 1.0 for all individual classes (meningioma, glioma, notumor, pituitary) and for the micro-average. This indicates that the model is able to distinguish between the different classes with almost perfect accuracy based on this metric. The curves are all hugging the top-left corner of the graph, which is the optimal position. The dashed black line represents a random classifier (AUC = 0.5).

ii. **Precision, Recall, and F1-score**

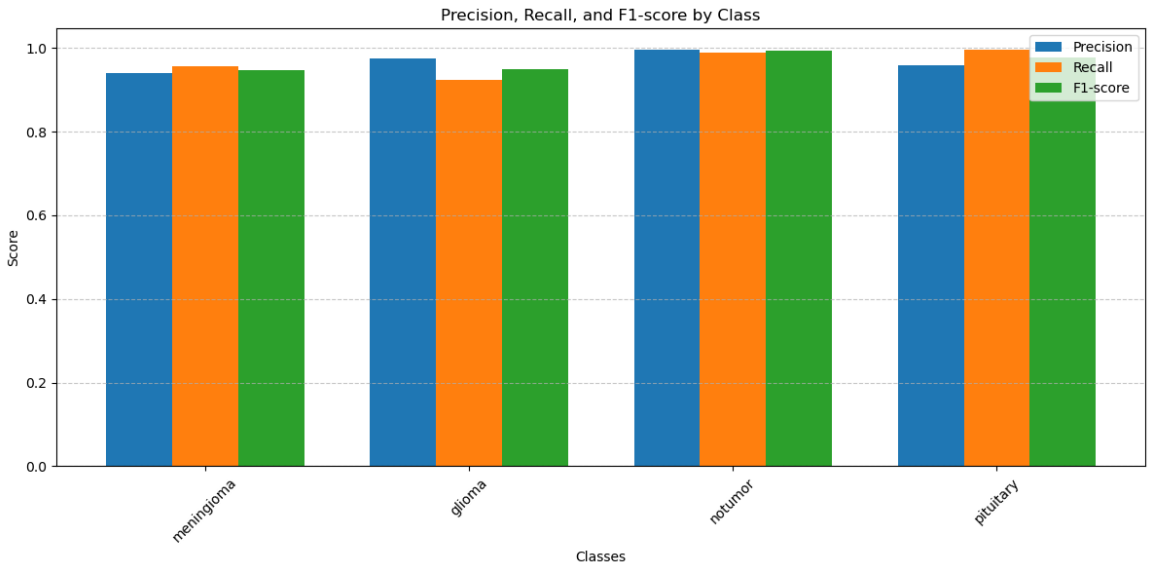


Figure 5.2: F1 Score, Precision, and Recall for VGG16 Model

(a) **Precision**: Our model has high precision for all classes (above 0.95 for most), with ”notumor” and ”pituitary” being particularly high, indicating very few false positives.

(b) **Recall**: Our model also shows high recall across all classes, particularly for ”notumor” and ”pituitary,” indicating it correctly identifies almost all instances of these classes.

(c) **F1-score**: The F1-scores are also very high (close to 0.98 for all classes), confirming the model’s balanced performance in identifying classes correctly and comprehensively.

iii. **Model Training History**

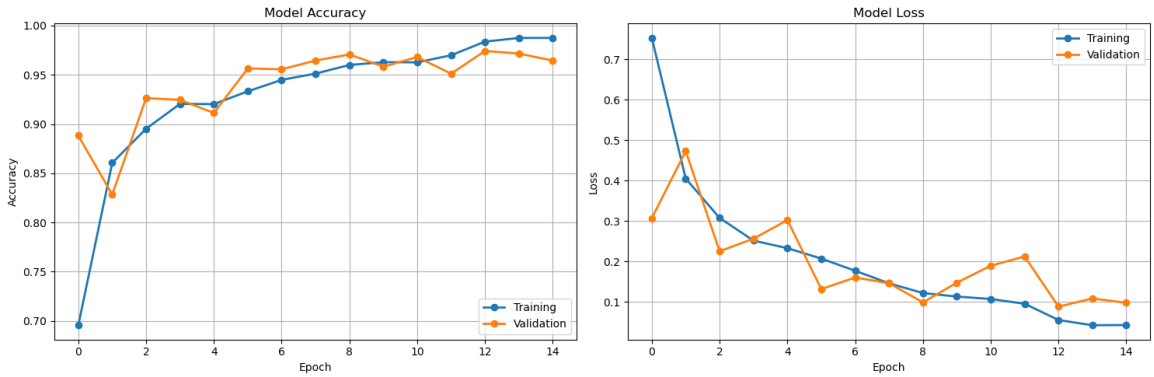


Figure 5.3: Model Training History

This image shows the model's performance during the training process over 14 epochs.

**Model Accuracy:** The left plot shows both training accuracy (blue line) and validation accuracy (orange line).

- The training accuracy steadily increases and reaches a high of about 99% by the final epoch.
- The validation accuracy also increases and closely follows the training accuracy, reaching a high of over 96%.
- The closeness between the training and validation accuracy curves indicates that the model is learning effectively and generalizing well to new, unseen data, with no significant signs of overfitting.

**Model Loss:** The right plot shows the training loss and validation loss.

- The training loss (blue line) decreases consistently throughout training, indicating the model is getting better at making correct predictions.
- The validation loss (orange line) also decreases, mirroring the training loss trend, which again suggests the model isn't overfitting. The validation loss shows some minor fluctuations, but the overall trend is a consistent decline.

iv. **Confusion Matrix**

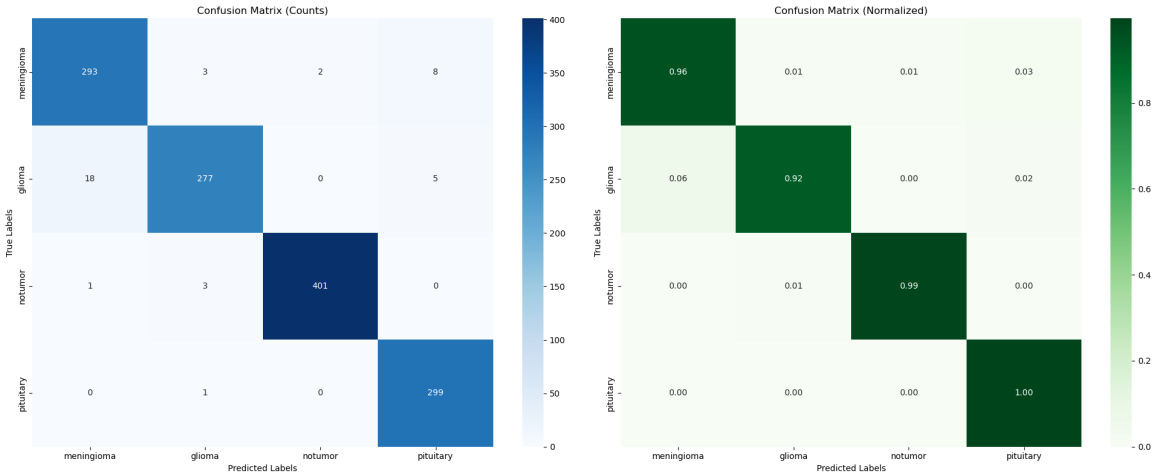


Figure 5.4: Confusion Matrix for VGG16 Model

The two heatmaps show the same data: the left one shows raw counts, and the right one shows normalized percentages.

- **Rows:** Represent the true labels (the actual class of the MRI).
- **Columns:** Represent the predicted labels (the class the model assigned).
- **Diagonal Elements:** The numbers along the main diagonal (e.g., 293 for meningioma, 277 for glioma) represent the number of correct predictions for each class.

- **Off-Diagonal Elements:** The numbers off the diagonal represent misclassifications.

**Count Matrix (Left):**

Table 5.6: Confusion matrix showing counts of correctly classified and misclassified MRI samples.

| True Class | Meningioma | Glioma | Notumor | Pituitary |
|---|---|---|---|---|
| Meningioma | 293 | 3 | 2 | 8 |
| Glioma | 18 | 277 | 0 | 5 |
| Notumor | 0 | 0 | 401 | 0 |
| Pituitary | 0 | 1 | 0 | 299 |

v. **Normalized Matrix (Right):**

- This heatmap shows the same data as percentages of the total for each true class.
- The highest misclassification rate appears to be glioma being misclassified as meningioma (6%), and meningioma being misclassified as pituitary (3%). These are minor errors given the overall high accuracy.

# CHAPTER 6
# CONCLUSION AND FUTURE RECOMMENDATIONS

## 6.1. Conclusion

The application provides an effective tool for analyzing brain MRI images and managing results. Users can upload images, receive predictions on tumor presence, and access past results along with profile management features. The frontend is built with React (Vite + Tailwind) served by Nginx, the backend uses Node/Express with MongoDB, and the ML model runs in Python. The system is thoroughly tested to ensure reliability and smooth deployment. The model demonstrates excellent performance, with almost all metrics above 0.97. Precision measures how many predicted positive cases were correct, recall measures how many actual positives were correctly identified, and F1-score balances precision and recall. Accuracy reflects overall correctness, support indicates sample counts per class, macro average treats all classes equally, and weighted average accounts for class size. Overall, the system achieves highly accurate tumor classification, making it a dependable tool for MRI analysis.

## 6.2. Future Recommendations

- **Expand Model Types:** Include additional brain tumor types or other neurological conditions to improve coverage.
- **Improve User Interface:** Enhance UI/UX for easier navigation, visualization of results, and accessibility.
- **Strengthen Security:** Implement advanced authentication, encryption, and secure data storage practices.
- **Automate CI/CD:** Set up continuous integration and deployment pipelines for faster updates and maintenance.
- **Integrate Cloud Deployment:** Deploy the system on a cloud platform for scalability and remote access.
- **Enhance ML Model:** Continuously retrain the model with more data to improve accuracy and robustness.
- **Add Reporting Features:** Provide detailed analytics and visual reports for medical professionals.

# REFERENCES

[1] The Preston Robert Tisch Brain Tumor Center, "How common is a brain tumor," 2024, accessed May 28, 2025. [Online]. Available: https://tischbraintumorcenter.duke.edu/blog/how-common-brain-tumor

[2] I. Ilic and M. Ilic, "International patterns and trends in the brain cancer incidence and mortality: An observational study based on the global burden of disease," *Heliyon*, vol. 9, no. 7, p. e18222, 2023.

[3] S. Mathivanan, S. Sonaimuthu, S. Murugesan *et al.*, "Employing deep learning and transfer learning for accurate brain tumor detection," *Scientific Reports*, vol. 14, p. 7232, 2024.

[4] B. B. Vimala, S. Srinivasan, S. Mathivanan *et al.*, "Detection and classification of brain tumor using hybrid deep learning models," *Scientific Reports*, vol. 13, p. 23029, 2023.

[5] M. Khaliki and M. Başarslan, "Brain tumor detection from images and comparison with transfer learning methods and 3-layer cnn," *Scientific Reports*, vol. 14, no. 1, p. 2664, 2024.

[6] F. Zohra *et al.*, "Image segmentation and classification using neural network," *arXiv preprint*, 2024, available at: https://arxiv.org/abs/placeholder.

[7] S. Srinivasan, B. B. Vimala, S. Mathivanan *et al.*, "A hybrid deep cnn model for brain tumor image multi-classification," *Scientific Reports*, vol. 14, p. 23029, 2024.

[8] S. Aksoy, "Brain tumor detection and classification via vgg16-based deep learning on mri imaging," in *Proceedings of [Conference Name]*, 2025, under review.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proceedings of the 32nd International Conference on Machine Learning*, pp. 448–456, 2015.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[12] JWT.io, "Introduction to json web tokens," 2024, accessed: 2025-09-03. [Online]. Available: https://www.jwt.io/introduction#what-is-json-web-token

[13] GeeksforGeeks, "Npm bcrypt," 2024, accessed: 2025-09-03. [Online]. Available: https://www.geeksforgeeks.org/node-js/npm-bcrypt/

# APPENDIX