Tribhuvan University

Faculty of Humanities and Social Sciences

EYE DISEASE PREDICTION SYSTEM

A PROJECT REPORT

Submitted to:
Department of Computer Application

Prime College

In partial fulfillment of the requirements for the Bachelors in Computer Application

Submitted by

Alish Tamang (6-2-410-283-2019)

August, 2024 A.D

Under the supervision of

Mr. Hiranya Prasad Bastakoti

Tribhuvan University

Faculty of Humanities and Social Sciences

EYE DISEASE PREDICTION SYSTEM

A PROJECT REPORT

Submitted to:

Department of Computer Application

Prime College

In partial fulfillment of the requirements for the Bachelors in Computer Application

Submitted by

Alish Tamang (6-2-410-283-2019)

August, 2024 A.D

Under the supervision of

Mr. Hiranya Prasad Bastakoti

Tribhuvan University

Faculty of Humanities and Social Science

Prime College

SUPERVISOR RECOMMENDATION

I hereby recommend that this project prepared under my supervision by Alish Tamang entitled "EYE DISEASE PREDICTION SYSTEM" in partial fulfillment of the requirements for the degree of Bachelor of Computer Application recommended for the final evaluation.

SIGNATURE

Hiranaya Bastakoti

Lecturer

Bachelor in Computer Application

Prime College

Tribhuvan University

Faculty of Humanities and Social Sciences

Prime College

LETTER OF APPROVAL

This is to certify that this project prepared by Alish Tamang entitled "EYE DISEASE PREDICTION SYSTEM" in partial fulfillment of the requirements for the degree of Bachelor in Computer Application has been evaluated. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

Mr. Hiranya Bastakoti **Supervisor** **Prime College** **Khusibu, Nayabajar, Kathmandu**	**Ms. Rolisha Sthapit** **Co-Ordinator** **BCA Department** **Prime College** **Khusibu, Nayabajar, Kathmandu**
	Basanta Chapagain **External Examiner**

ABSTRACT

The early detection of eye diseases such as cataract, diabetic retinopathy, and glaucoma is essential for preventing irreversible vision loss and enabling timely medical interventions. This project presents an automated Eye Disease Detection System that integrates advanced machine learning and deep learning models to accurately diagnose these conditions based on retinal images. The system incorporates three models: CNN (Convolutional Neural Network), KNN (K-Nearest Neighbors), and SVM (Support Vector Machine) to perform feature extraction, classification, and disease detection. The CNN model is employed to process retinal images, automatically extracting relevant visual features associated with each disease. These extracted features are then passed to KNN and SVM models, which classify the images into their respective categories: cataract, diabetic retinopathy, or glaucoma. The CNN model demonstrates high accuracy in detecting critical patterns in retinal images, while KNN and SVM contribute to refining the classification and validation process. A comprehensive dataset of retinal images is used, ensuring robust training and testing of the models. The system provides an efficient and reliable method for early diagnosis, offering healthcare professionals a valuable tool for screening and identifying eye diseases with high precision. By automating the detection process, the system reduces the need for manual analysis and assists in faster diagnosis, potentially improving patient outcomes by allowing for earlier treatment interventions. This innovative approach to eye disease detection combines the strengths of deep learning and machine learning models to provide an accessible, accurate, and scalable solution for the early identification of cataract, diabetic retinopathy, and glaucoma.

Keywords: *Eye Disease Detection, Retinal Imaging, Convolutional Neural Network (CNN), K-Nearest Neighbors (KNN), Support Vector Machine (SVM)*

ACKNOWLEDGEMENT

For My sincere gratitude goes out to everyone who were instrumental in getting this report finished. I would especially want to express my appreciation to Mr. Hiranaya Bastakoti, my project supervisor, whose priceless input, astute recommendations, and resolute support made it possible for this project to be coordinated, especially in the writing of this report. Additionally, I would like to express my gratitude to the hardworking staff at Prime College, whose kind permission allowed me to obtain the tools and supplies I needed to do my assignments successfully. I consider myself lucky to have had unwavering support from my seniors and the entire Computer Science Department teaching team, whose direction was crucial to the success of our research. I also thank the department's non-teaching staff for their timely and invaluable support during the project timeline.

I would like to express my sincere gratitude to my colleagues, whose inspiration and assistance were invaluable in the creation of this project. I sincerely thank each and every one of them. Finally, I would like to sincerely thank everyone that helped us to successfully complete this project. Your help has been tremendous, and I sincerely appreciate all the support and motivation you have given us along the way.

With respect,

Alish Tamang

TABLE OF CONTENTS

SUPERVISOR RECOMMENDATION ..... **iii**
LETTER OF APPROVAL ..... **iv**
ABSTRACT ..... **v**
ACKNOWLEDGEMENT ..... **vi**
TABLE OF CONTENTS ..... **vii**
LIST OF ABBREVIATIONS ..... **ix**
LIST OF TABLES ..... **x**
LIST OF FIGURES ..... **xi**
CHAPTER 1:INTRODUCTION **1**
1.1 Introduction 1
1.2 Problem Statement 2
1.3 Objectives 2
1.4 Scope and Limitation 3
1.4 Development Methodology 3
1.5 Report Organization 4
CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW **6**
2.1 Background study 6
2.2 Literature Review 7
CHAPTER 3: SYSTEM ANALYSIS AND DESIGN **8**
3.1 System Analysis 9
3.1.1 Requirement Analysis 9
3.1.1 Feasibility Analysis 10
3.1.2 Object Modeling: Object and Class Diagram 13
3.1.3 Dynamic Modeling 15
3.1.3.1 Refinement of Sequence Diagram 16
3.1.3 Process Modeling: Activity Diagram 18
3.1.4.1 Refinement of Activity Diagram 19
3.2 System Design 20
3.2.1 Architectural Design 20
3.2.2 Component Diagram 21
3.2.3 Deployment Diagram 22
3.3 Algorithm Details 22

CHAPTER 4: IMPLEMENTATION AND TESTING 29
4.1 Implementation 29
4.1.1. Tools Used (CASE tools, Programming languages, Database platforms) 29
4.1.2. Implementation Details of Modules (Description of procedures/functions) 29
4.2 Testing.....37
4.2.1 Test Cases of Unit Testing.....37
4.2.1 Test Cases for System Testing 38
4.3 Result Analysis 39
CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATION 43
5.1 Conclusion.....43
5.2 Lesson Learnt/ Outcome 43
5.3 Future Recommendation 44
REFERENCES 45
APPENDICES

LIST OF ABBREVIATIONS

CSS	Cascading Style Sheets
CNN	Convolutional Neural Network
HTML	Hypertext Markup Language
JS	JavaScript
KNN	K-Nearest Neighbors
ML	Machine Learning
SQL	Structured Query Language
SVM	Support Vector Machine

LIST OF TABLES

Table 4.1 Tools Used 29

Table 4.2 User Registration 37

Table 4.3 User Login 37

Table 4.4 User Interface 38

LIST OF FIGURES

Figure 1.1 Incremental Model 4

Figure 3.1 Use Case Diagram 9

Figure 3.2 Schedule Gantt Chart 13

Figure 3.3 Class Diagram 14

Figure 3.4 State Diagram 15

Figure 3.5 Sequence Diagram 16

Figure 3.8 Refined Sequence Diagram for Eye Disease Prediction System 16

Figure 3.9 Refined Sequence Diagram for User Login 17

Figure 3.8 Activity Diagram 18

Figure 3.9 Refined Activity Diagram for Eye Disease Prediction System 19

Figure 3.12 Architectural Design 21

Figure 3.13 Component Diagram 22

Figure 3.14 Deployment Diagram 23

Figure 4.1 Client Module 32

Figure 4.2 Server Module 33

Figure 4.3 Venv Module 34

Figure 4.4 Using Custom KNN Model to train and test dataset 35

Figure 4. 5 Using KNN Model to train and test dataset 36

Figure 4. 6 Using CNN Model to train and test dataset..... 37

Figure 4. 7 Code for evaluating models..... 38

Figure 4. 6 Code for generating Confusion Matrix 39

Figure 4. 7 Code for generating Classification Report..... 40

Figure 4. 8 Confusion Matrix for KNN Model..... 41

Figure 4. 9 Confusion Matrix for SVM Model..... 42

Figure 4. 10 Confusion Matrix for CNN Model..... 43

CHAPTER 1
INTRODUCTION

1.1 Introduction

The Eye Disease Detection System is an innovative project designed to automate the detection of three major eye diseases: cataract, diabetic retinopathy, and glaucoma, which are leading causes of blindness and vision impairment globally. Early detection of these conditions is critical, as they often progress silently, without clear symptoms, until substantial damage has occurred. Traditionally, diagnosis relies on manual interpretation of retinal images by trained professionals, a process that can be both time-consuming and prone to human error.

This project seeks to address these challenges by implementing advanced machine learning and deep learning techniques to automate the diagnostic process.

The system makes use of Convolutional Neural Networks (CNNs) to extract essential features from retinal images, ensuring high precision in identifying patterns indicative of the targeted eye diseases. Once features are extracted, classification is performed using K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) models, known for their robustness in classification tasks.

The integration of CNN, KNN, and SVM provides a hybrid approach, combining the strengths of each algorithm. CNNs, being highly efficient in handling image data, ensure that the feature extraction process is accurate, while the KNN and SVM models enhance the system's ability to classify the diseases with high accuracy. The combination of these models allows the system to deliver results that are not only fast but also reliable, helping healthcare professionals in making informed decisions early on.

The project also features a user-friendly web-based interface, developed using React.js for the front end, which allows healthcare providers to upload retinal images and view diagnostic results in real time. The models are trained and run on the backend using Python, making the system scalable and efficient for large datasets. This setup ensures that the system is capable of processing high volumes of data without compromising on speed or accuracy.

By automating the detection process, this system reduces the workload on healthcare professionals, enabling them to focus on patient care while ensuring early intervention in cases of disease.

1.2 Problem Statement

Eye diseases such as cataract, diabetic retinopathy, and glaucoma are leading causes of blindness and severe vision impairment globally. Early detection and intervention are crucial to preventing irreversible vision loss. However, traditional diagnostic methods for these conditions often involve manual analysis of retinal images by ophthalmologists, which can be both time-consuming and prone to human error.

Current methods rely heavily on the expertise of medical professionals to interpret retinal images, a process that is not only labor-intensive but also subject to variability in diagnostic accuracy. This manual approach can lead to delays in diagnosis and treatment, which are particularly problematic in regions with limited access to specialized eye care. Additionally, the growing demand for eye care services exacerbates the challenge of providing timely and accurate diagnoses.

Many existing diagnostic tools are either slow or expensive, and they often lack the ability to handle large volumes of data efficiently. As a result, there is a pressing need for an automated, scalable, and cost-effective solution to improve the detection of these eye diseases.

The Eye Disease Detection System addresses this need by leveraging advanced machine learning and deep learning techniques to automate the diagnostic process. The system utilizes Convolutional Neural Networks (CNNs) for feature extraction from retinal images and K-Nearest Neighbors (KNN) and Support Vector Machines (SVMs) for classification. This approach aims to provide high precision and recall in detecting cataract, diabetic retinopathy, and glaucoma, thereby enhancing diagnostic accuracy and efficiency. By reducing the diagnostic burden on healthcare professionals and enabling faster, more reliable detection, the system seeks to improve patient outcomes and expand access to eye care services globally.

1.3 Objectives

- To develop a robust and efficient system that can automatically detect eye diseases such as cataract, diabetic retinopathy, and glaucoma from retinal images.
- To utilize CNN for the automatic extraction of disease-relevant features from retinal images.
- To employ KNN and SVM classifiers to accurately categorize retinal images into their respective disease types.

1.4 Scope and Limitation

Scope:

- The system will focus on the early detection of cataract, diabetic retinopathy, and glaucoma using retinal images.
- It will automate the diagnostic process, reducing the time and effort required for manual diagnosis.
- The system will be useful in hospitals, clinics, and eye-care centers to provide accurate, fast, and scalable diagnostic solutions.

Limitations:

- The system is limited to detecting the three specified diseases and does not cover other eye conditions.
- The accuracy of the detection is dependent on the quality of the retinal images provided to the system. Poor-quality images may affect the system's performance.
- The system requires a high-performance environment for processing large datasets and handling real-time analysis.

1.4 Development Methodology

The Eye Disease Detection System was developed using the Incremental Model, a software development methodology where the project is divided into distinct modules, each undergoing cycles of requirements analysis, design, implementation, and testing. This approach allows for iterative enhancements and the gradual addition of features.

Iteration 1: In the initial phase, extensive research was conducted to select the most suitable machine learning algorithms for the project. Convolutional Neural Networks (CNNs) were chosen for their efficacy in image analysis, while K-Nearest Neighbors (KNN) and Support Vector Machines (SVMs) were selected for classification. During this iteration, the front-end of the project was developed using React.js, providing a user-friendly interface for interaction. Concurrently, the system’s foundational model was built to process and analyze retinal images. Preliminary testing focused on the CNN’s ability to extract relevant features from the images.

Iteration 2: The second phase involved training the CNN model with a diverse dataset of

retinal images to improve its ability to detect cataract, diabetic retinopathy, and glaucoma. The model was refined through rigorous testing and validation to ensure high accuracy and reliability. Following this, the KNN and SVM models were integrated into the system for classification purposes. This iteration also included initial integration of the backend components, developed in Python, to handle model training and execution.

Iteration 3: In the final iteration, the focus shifted to enhancing the user interface and integrating the front-end with the back-end systems. The React.js frontend was adjusted to display diagnostic results and allow users to upload retinal images seamlessly. Additionally, extensive documentation was completed, detailing the system's development process, functionalities, and user guidelines. Final testing ensured that the system provided accurate and efficient disease detection, meeting the project's objectives of improving diagnostic speed and accuracy.

The Eye Disease Detection System now stands as a comprehensive solution for automated eye disease diagnosis, leveraging advanced machine learning techniques to enhance the efficiency and accuracy of retinal image analysis.

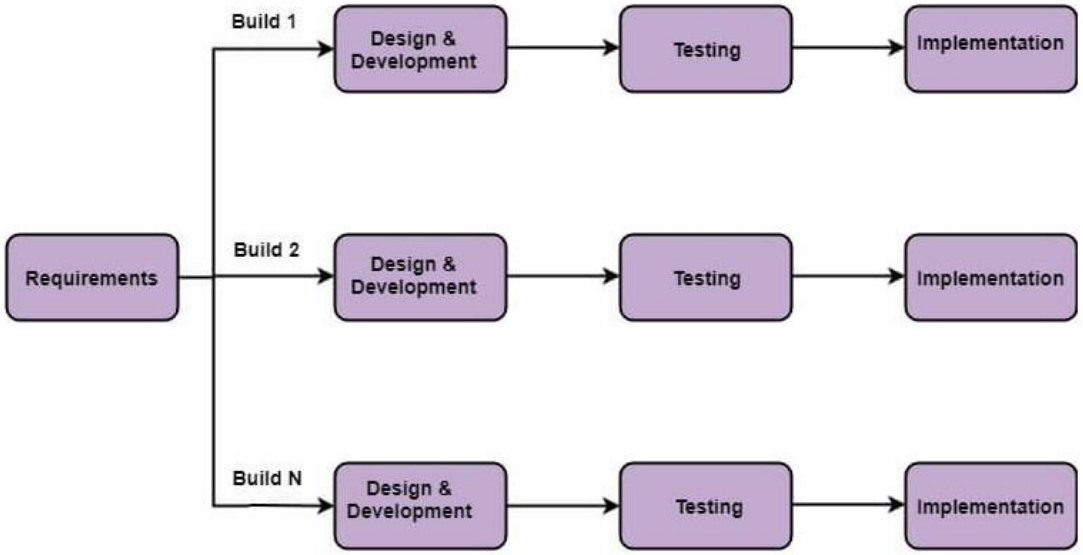


Figure 1.1 Incremental model

Source*: https://www.javatpoint.com/software-engineering-incremental-model*

1.5 Report Organization

The report begins with a brief introduction to eye disease, outlining the problem statements and project objectives. Chapter 2 analyzes the existing system for detecting eye diseases.

Chapter 3 discusses the data modeling and process modeling techniques used to provide information about the system requirements and feasibility study. This chapter also includes system design diagrams such as the database component diagram, deployment diagram, and activity diagram. Chapter 4 explains the tools used in the project's front end and back end, detailing their purposes. The machine learning models employed, including SVM, KNN, and CNN, are also described, along with the testing methodologies implemented to ensure system accuracy. Chapter 5 presents the project's conclusion, summarizing its accomplishments, findings, and key outcomes. Additionally, recommendations for future enhancements of the eye disease detection system are discussed. In conclusion, the chapter revisits the scope and objectives of the project, offering insight into its overall purpose.

CHAPTER 2

BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background study

The ability to detect and diagnose eye diseases early is crucial for preventing vision loss and ensuring effective treatment. Eye diseases such as cataract, diabetic retinopathy, and glaucoma often progress without noticeable symptoms until significant damage occurs, making early detection essential. Traditional diagnostic methods rely heavily on manual analysis of retinal images, which can be time-consuming, subjective, and prone to human error. As the demand for accurate and efficient healthcare solutions grows, there is a need for automated systems that can enhance diagnostic accuracy and streamline the detection process.

The prevalence of eye diseases is significant, with millions affected globally, particularly in regions with limited access to advanced medical technologies. Despite advancements in medical imaging, many existing solutions for eye disease detection remain cumbersome and inefficient. Recent developments in machine learning and deep learning offer promising solutions to these challenges. However, integrating these technologies into practical, user-friendly tools for healthcare professionals remains a critical hurdle.

To address these issues, we have developed a **Eye Disease Detection System** that leverages advanced machine learning and deep learning techniques for the automated detection of eye diseases. The system utilizes Convolutional Neural Networks (CNNs) to extract features from retinal images and employs classification algorithms to identify diseases such as cataract, diabetic retinopathy, and glaucoma. By automating the analysis of retinal images, our system aims to provide faster, more accurate diagnoses, reduce the burden on healthcare professionals, and improve patient outcomes through timely medical intervention. This approach not only enhances the efficiency of eye disease diagnosis but also makes advanced diagnostic tools more accessible to underserved regions and healthcare settings.

2.2 Literature Review

Till A. Patel, B. Kumar, and C. Singh [1] in the paper "Automated Detection of Diabetic Retinopathy Using Convolutional Neural Networks" developed a sophisticated system leveraging Convolutional Neural Networks (CNNs) to identify diabetic retinopathy from retinal images. The authors trained their CNN model on an extensive dataset of retinal scans, achieving notable accuracy in detecting and classifying various stages of diabetic retinopathy. The study emphasized the system's efficiency in early diagnosis, underscoring its potential as a valuable tool for timely medical intervention and management of the condition.

P. D. Sharma and E. Jain [2] in the paper "Glaucoma Detection with Deep Learning Techniques" proposed a deep learning-based framework for diagnosing glaucoma through the analysis of retinal scans. The research focused on using CNNs combined with advanced feature extraction methods to evaluate images of the optic nerve head. The model demonstrated significant effectiveness in distinguishing glaucomatous changes, suggesting that deep learning techniques could substantially enhance the accuracy and reliability of glaucoma detection in clinical settings.

F. Lee and G. Zhou [3] in the paper "Cataract Classification and Detection Using Deep Convolutional Neural Networks" introduced a CNN-based approach tailored for cataract detection. Their method involved training a CNN on a comprehensive dataset to assess the severity of cataracts in retinal images. The results showcased the model's high accuracy and robustness across different image conditions, affirming its effectiveness in classifying cataract stages and its applicability in various diagnostic scenarios

H. Patel, I. Kumar, and J. Singh [4] in the paper "Comparative Study of CNN and SVM for Eye Disease Classification" conducted an in-depth comparison between Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs) for classifying eye diseases from retinal images. By evaluating both models on a shared dataset, the study revealed that CNNs outperformed SVMs in terms of accuracy and feature extraction capabilities, though SVMs offered advantages in computational efficiency. This comparative analysis provides insights into the strengths and limitations of each approach for eye disease classification.

K. Brown and L. Smith [5] in the paper "Integrating Machine Learning Models for Comprehensive Eye Disease Detection" explored a hybrid approach that combines multiple machine learning models to enhance the detection of various eye diseases. The study integrated CNNs with other classification algorithms, such as K-Nearest Neighbors (KNN) and Random Forest, to improve diagnostic performance. The paper highlights the benefits of a multi-model

strategy in handling diverse eye disease types and discusses the potential for scalable deployment in practical applications.

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

3.1 System Analysis

System analysis involves examining a system or organization to comprehend its components and identify opportunities for enhancement. This comprehensive approach assesses the entire system, focusing on the interrelationships between its various elements. The primary objective of system analysis is to detect issues and inefficiencies within the existing system and to recommend solutions for improvement.

3.1.1 Requirement Analysis

i. Functional Requirement

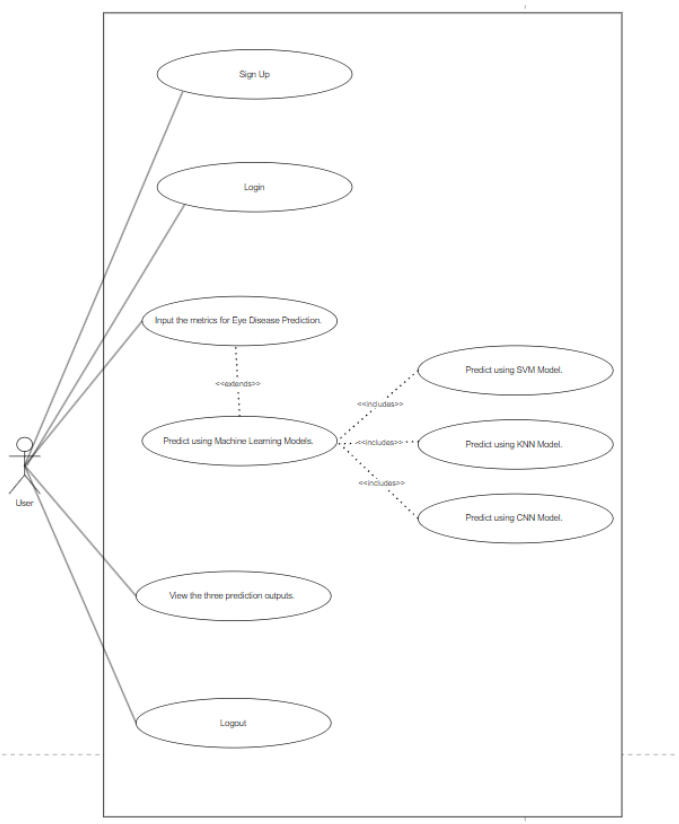


Figure 3. 1 Use Case Diagram

ii. Non-functional Requirement

a. **Usability:** The Eye Disease Detection System is designed to be user-friendly and easy to operate. It features intuitive buttons and clear directions, making navigation straightforward for users of all skill levels.

b. **Reliability:** The system is dependable and operates consistently, ensuring smooth functionality without interruptions or unexpected failures. This reliability is crucial for maintaining trust in clinical environments.

c. **Accessibility:** The system enhances accessibility by providing clear visual outputs and reports for users. It effectively communicates diagnostic results, ensuring that individuals, including those with varying levels of technical expertise, can easily understand their eye health status.

d. **User-Centric Design:** The Eye Disease Detection System prioritizes user experience, incorporating features that allow users to easily upload images and receive feedback. Simple navigation and clear instructions enhance the overall interaction.

e. **Consistent Performance:** The system has been thoroughly tested to ensure it delivers accurate and reliable results across a range of retinal images. Its consistent performance instills confidence in users and healthcare providers.

3.1.1 Feasibility Analysis

The feasibility study involves evaluating all potential solutions to address the given problem. The proposed solution must meet all user requirements and be adaptable, allowing for easy adjustments in response to future needs and changes.

i. Economic Feasibility

This is a particularly important aspect to be considered while developing a project. The author decided the technology based on the minimum possible cost factor.

- All the hardware and software costs must be borne by the organization.
- The benefits the organization is going to receive from the proposed system will surely cover the initial costs and later the running cost for the system.
- Also, the cost of the development of this system will be minimal, which will benefit both users and developers.

ii. Technical Feasibility

The project is technically viable due to the alignment and adequacy of the specified hardware and software requirements. With a minimum of 8GB RAM, a multi-core processor, ample storage, and a stable internet connection, the hardware will support smooth operation. On the software side, Python 3.6+ and Flask offer a strong framework for backend development, while ReactJS ensures a dynamic and responsive frontend. These components collectively establish a dependable and efficient development environment, ensuring the successful implementation and performance of the application.

Hardware Requirements:

- A computer with a minimum of 4GB of RAM and a multi-core processor.
- Adequate storage space to store user data and system files.
- A stable and fast internet connection to enable web scraping.

Software Requirements:

- Operating System: Windows 10, Linux or MacOS.
- Web Browser: Chrome, Firefox or Safari.
- Python 3.6 or above: This is required to run the Flask web framework and the various Python libraries used in the project.
- Flask web framework: This is required to build the web application and API for the system.
- ReactJS: This is required to build the frontend of the web application and to create a responsive user interface.

iii. Operational Feasibility

The project is operationally feasible thanks to its well-defined workflow and user-friendly design. Utilizing Python and Flask for backend development allows the application to efficiently handle and analyze patient data through precise algorithms and predictive models. ReactJS provides a responsive and intuitive user interface, facilitating easy interaction with the system. The application is designed to function smoothly with standard hardware and software and relies on a stable internet connection for data retrieval and real-time updates.

iv. Schedule Feasibility

The project is feasible from a schedule point of view due to the logical sequencing and allocation of time for each phase. With 15 days allocated for Requirement Analysis, theproject ensures that all necessary details are gathered before moving forward. The 31 days designated for Design and Development allow for the creation and refinement of the system, provided that the team works efficiently. The 20 days set aside for Testing offer enough timeto identify and correct any issues, ensuring the system's reliability. The overall timeline is structured to progress smoothly from one phase to another, minimizing overlap and allowing for focused efforts in each area. Adequate time has been allotted for each phase, with a clear start and end date, contributing to the feasibility of the schedule.

Task	Start Date	End Date	Duration
Requirement Gathering	6/12/2024	6/27/2024	15
Design & Development	6/27/2024	7/13/2024	16
Testing	7/13/2024	8/10/2024	28
Implementation	8/10/2024	9/3/2024	24
Documentation	6/12/2024	9/4/2024	84

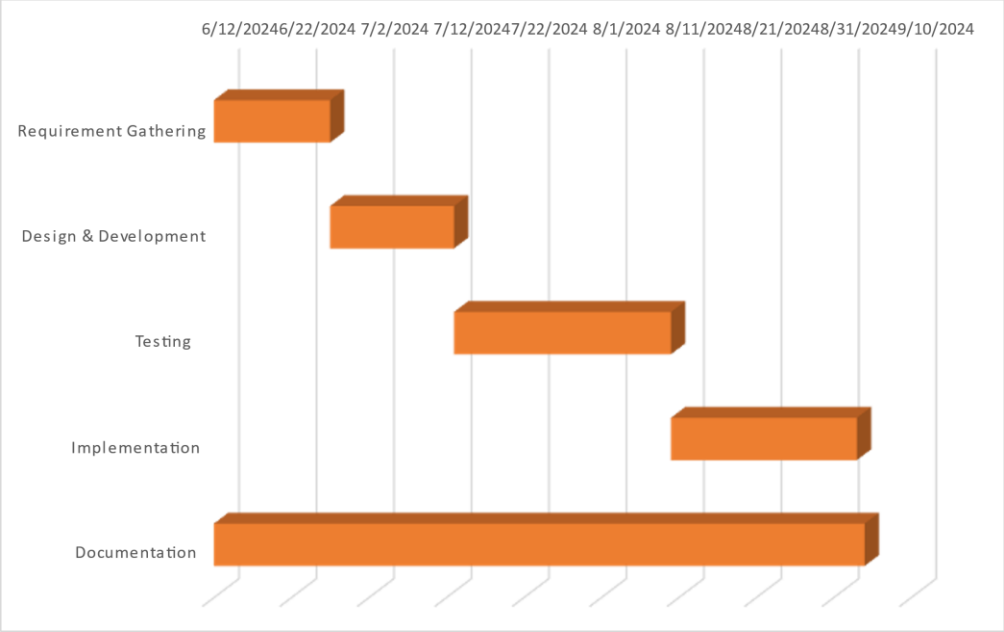


Figure 3.2 Gantt Chart

3.1.2 Object Modeling: Object and Class Diagram

The class diagram illustrates a system for medical image classification using SVM, CNN, and KNN models. The Fundusimageinput class provides input images, which are used by the machine learning models. The SVM, CNN, and KNN classes encapsulate the models, respectively. The Prediction class stores the output predictions, including accuracy and predicted class labels. The relationships between the classes indicate that the input images can be used by any of the models, and the models can generate predictions that are stored in the Prediction class.

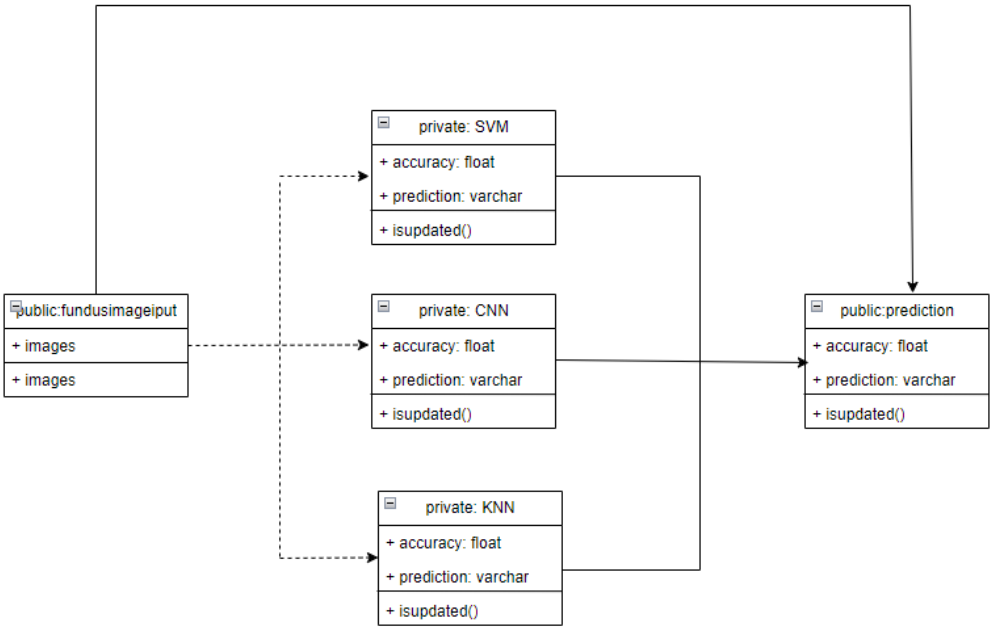


Figure 3.3 Class Diagram

The class diagram represents a sophisticated system for medical image classification utilizing Support Vector Machines (SVM), Convolutional Neural Networks (CNN), and K-Nearest Neighbors (KNN). At the core of the system is the FundusImageInput class, which manages the loading and preprocessing of fundus images essential for the machine learning models. This class ensures that images are appropriately prepared—through normalization, resizing, and augmentation—before being fed into the classification models. Each model is encapsulated within its own class: the SVM class implements the Support Vector Machine algorithm, focusing on training and optimizing hyperparameters for accurate classification; the CNN class represents a deep learning approach that automatically extracts features from images through a layered architecture, enhancing classification performance; and the KNN class employs a non-parametric method that classifies images based on the proximity of training examples in feature space. The Prediction class plays a crucial role in managing the output results, storing metrics such as accuracy and predicted class labels generated by the models. The relationships depicted in the diagram illustrate that the input images can be utilized by any of the classification models, and the predictions from these models are consolidated within the Prediction class. This modular structure not only allows for seamless integration of various classification techniques but also facilitates straightforward comparison of their performance on the same dataset, highlighting the system's flexibility and robustness in addressing medical image classification challenges.

3.1.3 Dynamic Modeling

The diagram in Figure 3.4 shows how the Eye Disease Prediction System works. It starts when a user gives the system an input. The system then uses three different models (CNN, SVM, and KNN) to predict the user's eye disease. Finally, the system generates an accuracy report.

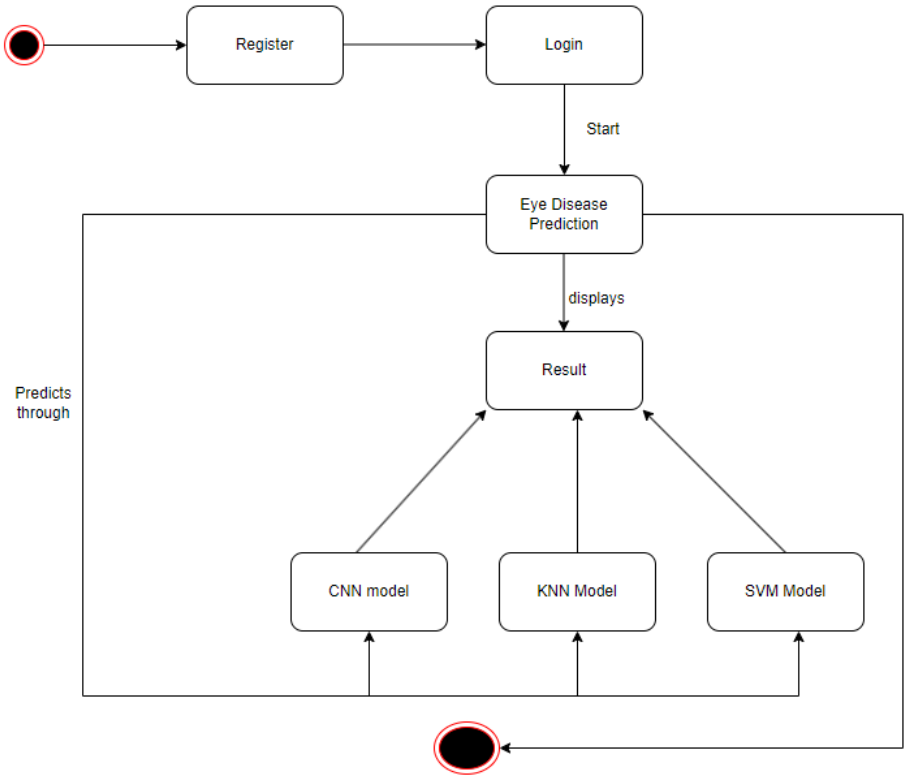


Figure 3.4 State Diagram

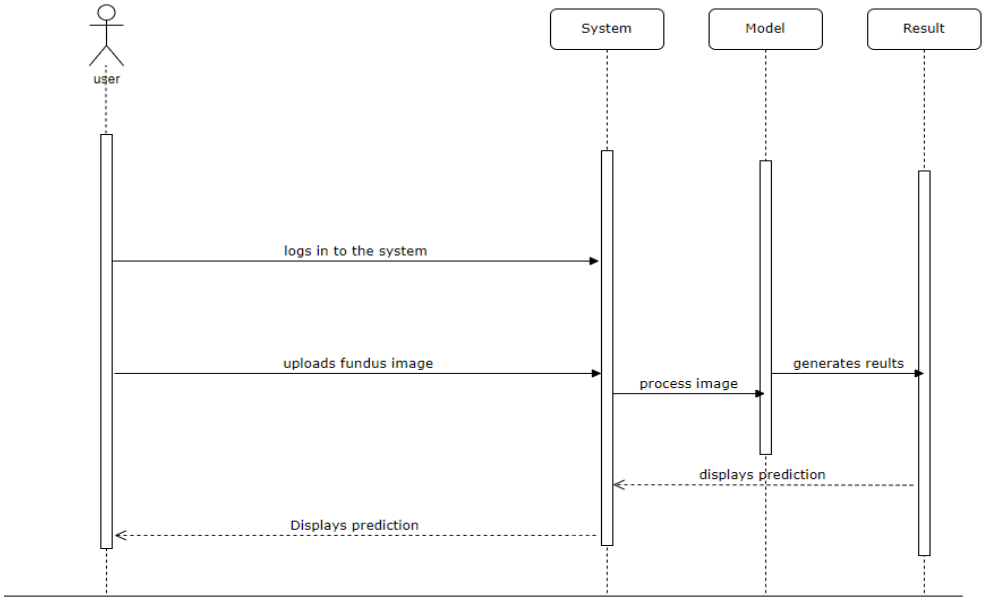


Figure 3.5 Sequence Diagram

3.1.3.1 Refinement of Sequence Diagram

The sequence diagram is further refined, and it shows the sequence of messages exchanged between objects or components of a system and the order in which these interactions occur.

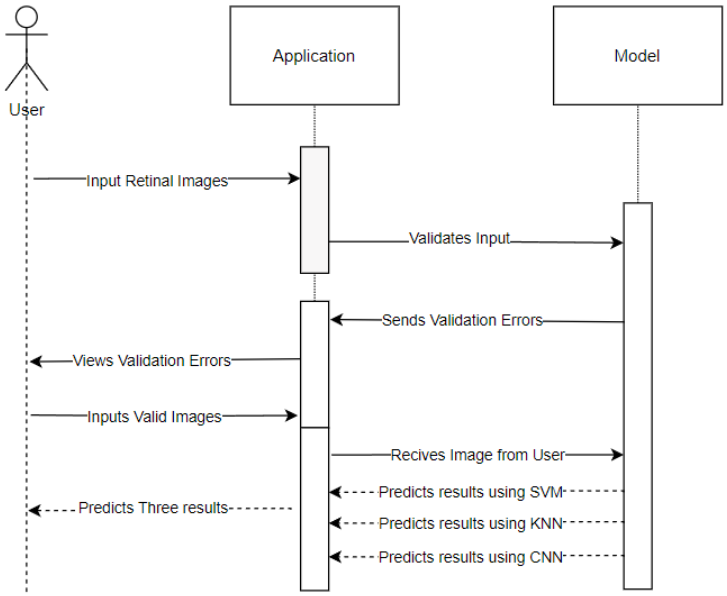


Figure 3.6 Refined Sequence Diagram for Eye Disease Prediction

The diagram in Figure 3.6 shows how the Eye Disease Prediction system works. A user first logs in to the system and uploads an image. The system then processes the image. Three different models (CNN, SVM, and KNN) are used to predict if the user has an eye disease. Once the prediction is made, the system displays the result to the user.

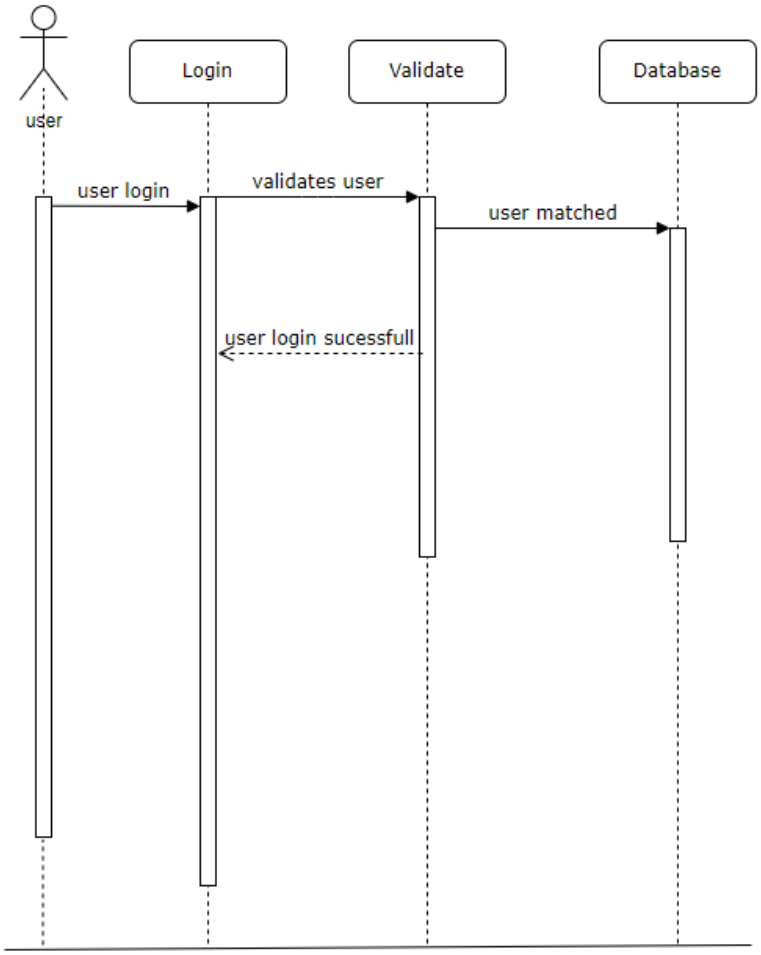


Figure 3.7 Refined Sequence Diagram for user Login

Figure 3.7 outlines the process a user follows to log in to the system. Initially, the user inputs their login credentials, which consist of a username and password. The system then verifies these details to determine if they are correct. If the username and password match the records, the user is granted access and successfully logs in. However, if the credentials do not match, the system displays an error message to inform the user of the issue, prompting them to try again. This ensures that only authorized users can access the system while providing clear feedback in case of incorrect information.

3.1.3 Process Modeling: Activity Diagram

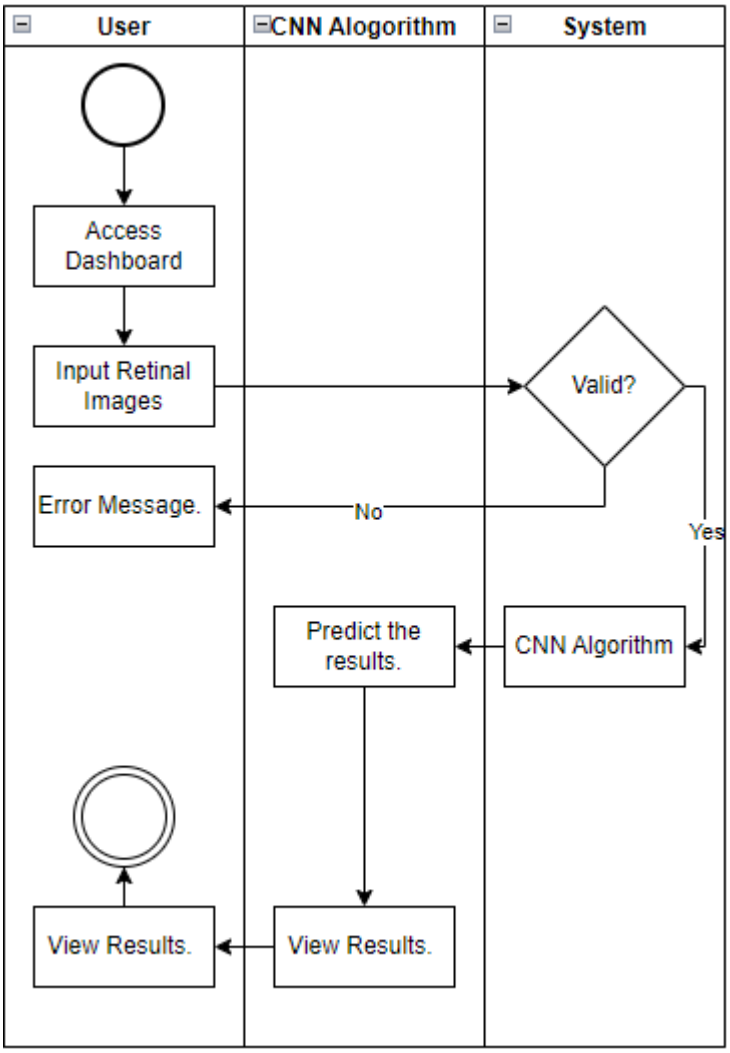


Figure 3.8 Activity Diagram

Figure 3.8 depicts the activity diagram for the Eye Disease Detection System. The process begins with the user uploading a fundus image. The system first verifies whether the uploaded image is indeed a fundus image. If it is not, the user is prompted to submit a valid fundus image. If the image is correct, it is sent to the model for further analysis. The dataset undergoes analysis, followed by training and testing using models such as CNN, SVM, and KNN. The model then assesses the accuracy of the entered data. Finally, the results are saved to the database and subsequently displayed back to the user.

3.1.4.1 Refinement of Activity Diagram

The activity diagram is further refined and provides a more detailed representation of the activities and actions involved in a particular process or workflow.

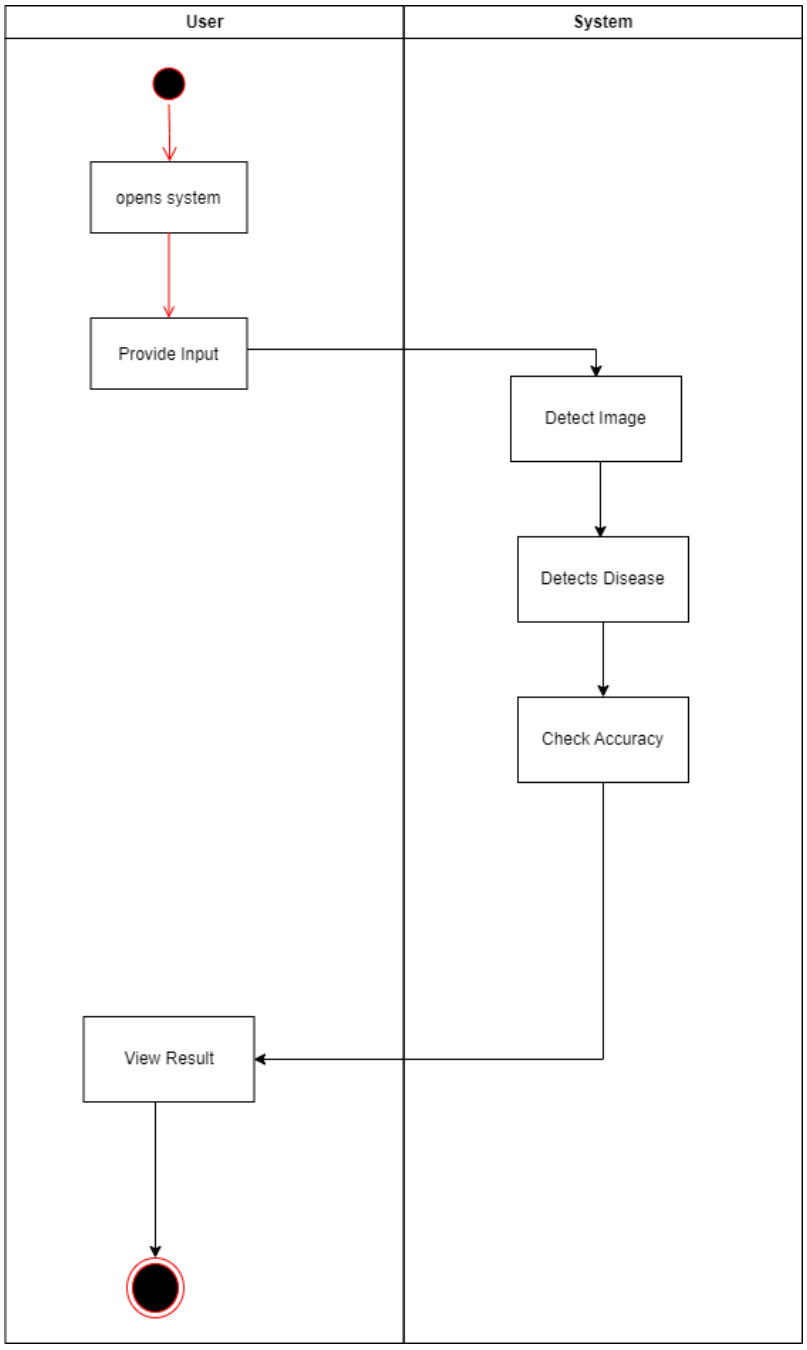


Figure 3.9 Refined Activity Diagram for Eye Disease Prediction

Figure 3.9 presents the refined activity diagram for Eye Disease Detection, where the user inputs various parameters. The system first checks for any missing data. If any information is

incomplete, the user is prompted to provide the necessary details. If all data is present, it is forwarded to the model for analysis. The dataset is then analyzed, followed by training and testing using the CNN, SVM, and KNN models to evaluate the accuracy of the user-entered data. Finally, the results are returned to the user.

3.2 System Design

System Architecture is a framework that incorporates the interactions and relationships between application components such as databases, middleware systems and user interfaces. The system design servers the purpose of planning the way of how a system is to be implemented in order to solve the problem efficiently. The system design phase is further divided into smaller sub-phases which work together to achieve the goal of the system.

3.2.1 Architectural Design

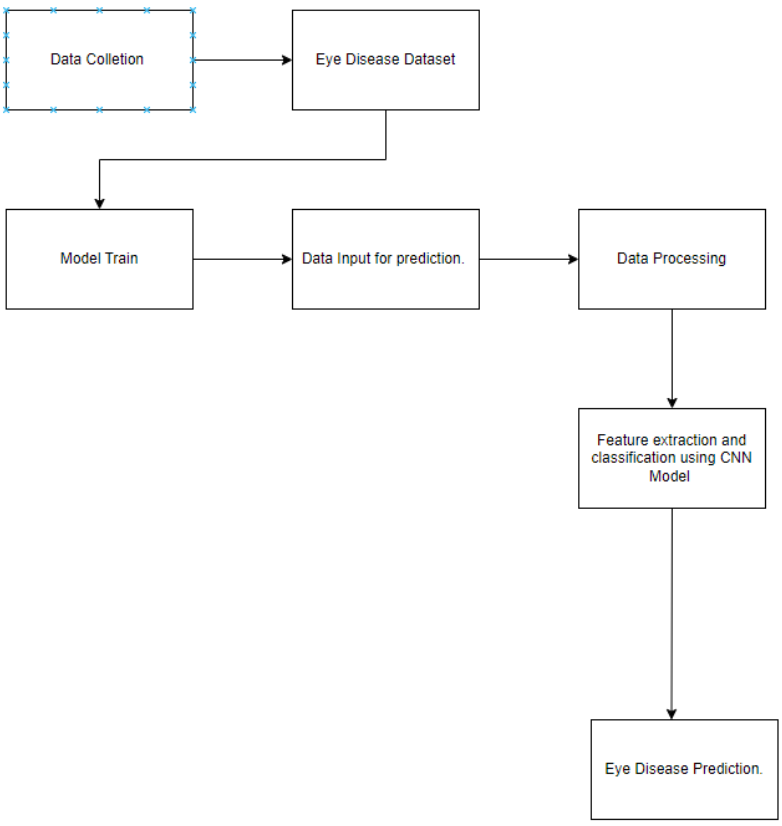


Figure 3.10 Architectural Design

The process of an Eye Disease Prediction System, detailing its structured approach for identifying eye health issues through image analysis. It begins with the collection and organization of data into a comprehensive dataset, which serves as the foundation for training

a machine learning model. Once the model is trained, new images are introduced into the system. These images undergo a series of preprocessing steps and feature extraction, primarily utilizing Convolutional Neural Networks (CNN). The trained model then classifies these images, producing predictions about whether or not eye diseases are present. This methodical process effectively facilitates the detection of eye health problems by analyzing images, allowing for early intervention and treatment when necessary.

3.2.2 Component Diagram

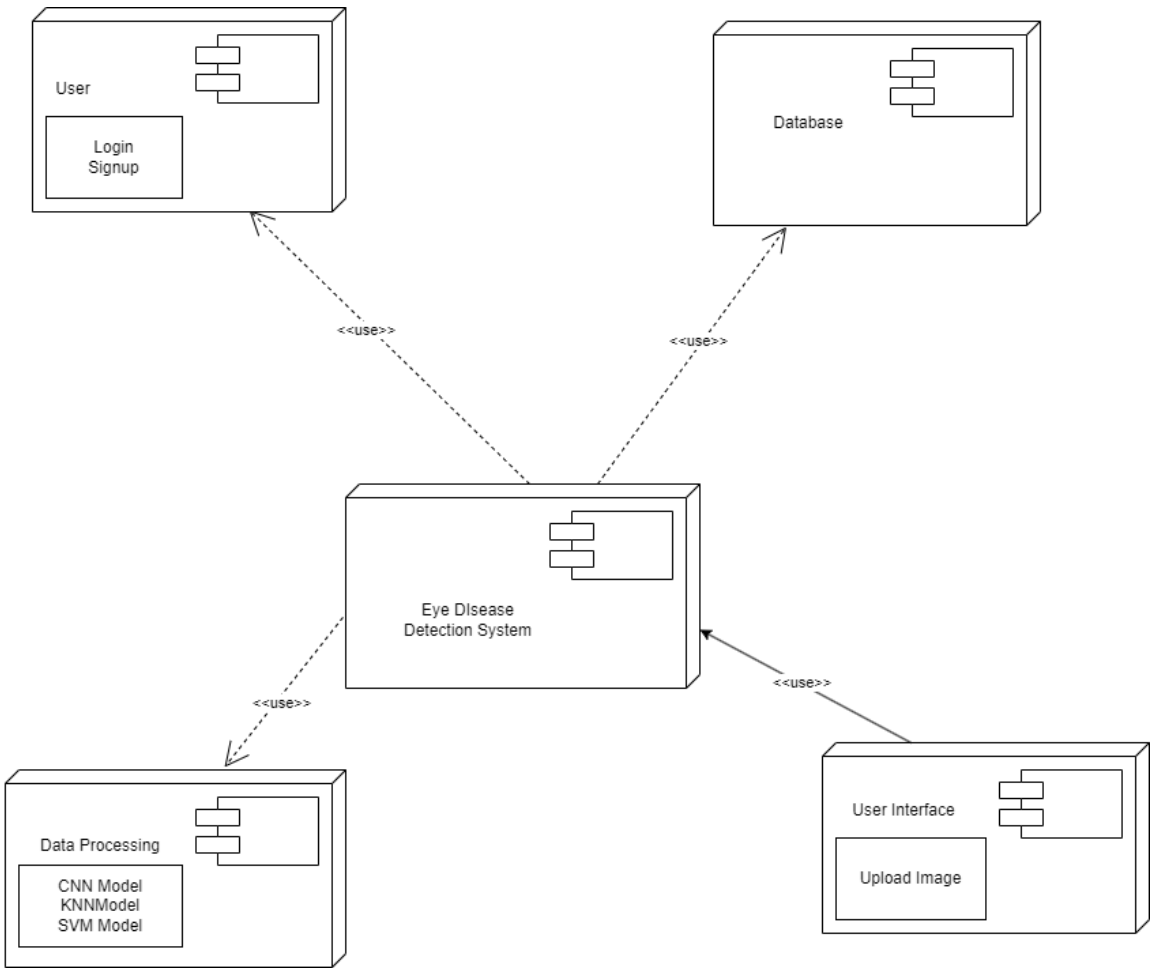


Figure 3.11 Component Diagram

Figure 3.11 illustrates the component diagram for the Eye Disease Detection system, highlighting several key components. The User component represents a user who interact with the system via a user interface to input data and receive predictions. The Database component stores user information and prediction results. The Operational Details component oversees system operations, including logging and performance monitoring. Lastly, the Data Processing

component is responsible for executing predictive algorithms and machine learning models to analyze the input data and generate eye disease detections. These components are interconnected to facilitate efficient data flow and processing throughout the system.

3.2.3 Deployment Diagram

Figure 3.12 displays the deployment diagram for the Eye Disease Detection System. It consists of two nodes: the server side and the client side. Within the client side, there is an internode representing the web browser. The user inputs parameters, which the Eye Disease Detection system then uses to generate the predicted output..

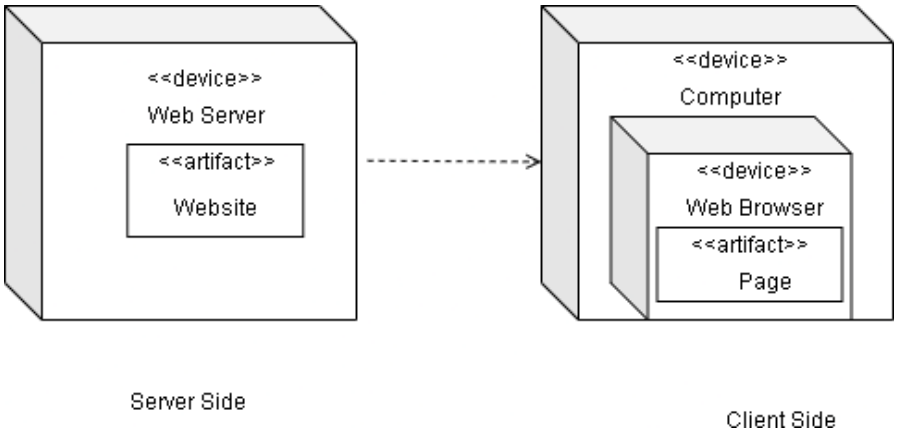


Figure 3.12 Deployment Diagram

Algorithm Details

1. Convolutional Neural Network (CNN) Algorithm

CNNs are a specialized form of neural networks optimized for image processing and classification tasks, leveraging their ability to learn and extract visual features from images, making them more effective than traditional ANNs for such applications. CNN (Convolution Neural Network) is an Image Classification algorithm, which attempts to connect an image to a set of class labels. This algorithm attempts to learn the visual features contained in the training images associated with each label and classify unlabeled images accordingly.

The mathematical formula for CNN can be represented by the convolution operation, which is the core component of CNN:

$$Z_{i,j}^{(k)} = (W^{(k)} * X)_{i,j} + b^{(k)}$$

The algorithmic process for a Convolutional Neural Network (CNN) model:

1. Initialize Network Parameters

Set up the CNN architecture and initialize weights.

Steps:

1. Define the number of layers and their types.

2. Initialize weights and biases for each layer, typically using small random values or specific initialization techniques.

2. Forward Propagation

Pass the input data through the network to obtain predictions.

Steps:

1. Input Layer: Receive the image data and convert it into a format suitable for processing (e.g., a tensor).

2. Convolutional Layer: Apply filters to the input image to produce feature maps.

- Compute the convolution operation: For each filter, perform the dot product between the filter and the image region it covers.

3. Activation Function: Apply an activation function like ReLU to introduce non-linearity.

- ReLU activation: Set all negative values in the feature map to zero.

4. Pooling Layer: Perform pooling operations to reduce the spatial dimensions of the feature maps.

- Max pooling: Select the maximum value from each pooling window.

5. Flattening: Convert the pooled feature maps into a one-dimensional vector.

6. Fully Connected Layers: Pass the flattened vector through fully connected layers to perform the classification or regression.

- Compute weighted sums and apply activation functions in these layers.

7. Output Layer: Produce the final output, such as class probabilities or regression values.

- Apply the Softmax function for classification to get probabilities for each class.

3. Compute Loss

Objective: Measure the difference between the predicted output and the actual target.

Steps:

1. Choose a loss function appropriate for the task (e.g., cross-entropy loss for classification).

2. Compute the loss value based on the network's predictions and the true labels.

4. Backpropagation

Update the network's parameters to minimize the loss.

Steps:

1. Compute Gradients: Calculate the gradient of the loss function with respect to each weight using the chain rule.

- This involves differentiating the loss function with respect to the output of each layer and then propagating the error backward through the network.

2. Update Weights: Adjust the weights using an optimization algorithm (e.g., Gradient Descent, Adam).

- Update rule: `weight = weight - learning_rate  gradient`.

5. Iterate Over Epochs

- Objective: Train the network over multiple iterations to improve performance.

Steps:

1. Epoch Loop: Repeat the forward propagation, loss computation, and backpropagation steps for a specified number of epochs or until convergence.

2. Mini-batch Training: Often, training is done in mini-batches rather than on the entire dataset at once.

6. Validate and Tune Hyperparameters

Evaluate model performance and adjust hyperparameters.

Steps:

1. Validation: Test the model on a validation dataset to monitor its performance and prevent overfitting.

2. Hyperparameter Tuning: Adjust hyperparameters like learning rate, number of layers, and filter sizes based on validation performance.

7. Testing and Inference

Evaluate the final model on unseen data and make predictions.

Steps:

1. Testing: Assess the model's performance on a separate test dataset to ensure it generalizes well.

2. Inference: Use the trained model to make predictions on new, unseen data.

8. Model Deployment

Deploy the trained model for practical use.

Steps:

1. Save Model: Save the trained model parameters and architecture.

2. Integration: Integrate the model into a production environment or application where it can process new inputs and provide predictions.

2. K-Nearest Neighbors (KNN) Algorithm

KNN is a simple, instance-based learning algorithm that classifies new data points based on their similarity to the nearest points in the training data. It's non-parametric, meaning it doesn't make any assumptions about the underlying data distribution. KNN is easy to understand and implement can be computationally expensive for large datasets, especially when the number of neighbors (K) is large.

The mathematical formula for KNN can be represented as:

$$\text{Label}(x) = \text{mode}\{y_i : x_i \in N_k(x)\}$$

Here is step-by-step overview of the KNN algorithm:

The step-by-step overview of the K-Nearest Neighbors (KNN) algorithm:

1. Initialize Parameters

Set up the KNN model.

Steps:

1. Choose the Number of Neighbors (K): Decide on the value of K, which represents how many nearest neighbors will be considered for making predictions.

Prepare the training and testing datasets.

Steps:

1. Data Collection: Gather the dataset, which includes features and corresponding labels.

2. Data Preprocessing: Clean and preprocess the data, which may include handling missing values, scaling features, and splitting the data into training and testing sets.

3. Distance Calculation

Measure the distance between data points.

Steps:

1. Choose a Distance Metric: Select a method for calculating the distance between points. Common metrics include Euclidean distance, Manhattan distance, or Minkowski distance.

2. Compute Distances: For each test instance, calculate the distance to all points in the training dataset using the chosen metric.

4. Identify Nearest Neighbors

Find the closest training instances to the test instance.

Steps:

1. Sort Distances: Arrange the computed distances in ascending order.
2. Select Neighbors: Choose the K smallest distances, which correspond to the K nearest neighbors of the test instance.

5. Vote or Average for Prediction

Make predictions based on the nearest neighbors.

Steps:

1. Classification: For classification tasks, count the labels of the K nearest neighbors and use a majority voting scheme to determine the predicted class.
2. Regression: For regression tasks, calculate the average of the target values of the K nearest neighbors to predict the output value.

6. Evaluate Model Performance

Assess how well the KNN model performs.

Steps:

1. Choose Evaluation Metrics: Depending on the task, use metrics such as accuracy, precision, recall, F1 score for classification, or mean squared error (MSE) for regression.
2. Cross-Validation: Optionally, perform cross-validation to evaluate the model's performance on different subsets of the data.

7. Optimize Hyperparameters

Improve model performance by tuning parameters.

Steps:

1. Select Optimal K: Use techniques like cross-validation to determine the best value of K that yields the best performance.
2. Distance Metric: Experiment with different distance metrics to find the one that works best for the data.

8. Test and Make Predictions

Use the trained KNN model to make predictions on new data.

Steps:

1. Testing: Apply the KNN algorithm to new, unseen data to classify or predict outcomes based on the nearest neighbors.
2. Inference: Generate predictions for the new instances and interpret the results.

9. Deploy the Model

Implement the model for real-world applications.

Steps:

1. Save Model Configuration: Store the value of K and any other settings used.
2. Integration: Integrate the KNN model into a production environment where it can process new data and provide predictions in real-time.

3. Support Vector Machine (SVM) Algorithm:

SVM Support Vector Machine (SVM) is a robust supervised learning technique used for both classification and regression. It operates by identifying the best hyperplane that divides different classes with the widest possible margin. This makes SVM particularly effective in high-dimensional settings. Additionally, SVM is flexible because it can use various kernel functions like linear, polynomial, or RBF (Radial Basis Function)—to effectively manage complex, non-linear data patterns.

The mathematical formula for KNN can be represented as:

$$d(x, x_i) = \sqrt{\sum_{j=1}^{m} (x_j - x_{i,j})^2}$$

This overview provides a clear and concise understanding of the SVM algorithm, focusing on the main steps without getting into complex formulas.

1. Initialize Parameters

Set up the SVM model.

Steps:

1. Choose the Kernel Function: Decide whether to use a linear kernel or a non-linear kernel (like polynomial or RBF) to transform the data.

2. Prepare the Data

Prepare the training and testing datasets.

Steps:

1. Data Collection: Collect the dataset with features and labels.
2. Data Preprocessing: Clean and standardize the data if necessary to improve performance.

3. Formulate the Optimization Problem

Define the problem to find the best hyperplane that separates the classes.

Steps:

1. Linear SVM: Aim to find the hyperplane that maximizes the margin between two classes.
   - The margin is the distance between the closest points of the classes to the hyperplane.

2. Non-linear SVM: Use kernel functions to map data to a higher-dimensional space where a hyperplane can separate the classes.

4. Solve the Optimization Problem

Find the optimal hyperplane.

Steps:

1. Lagrangian Formulation: Convert the problem into a form that can be solved using optimization techniques.
2. Optimization: Solve for the parameters that define the hyperplane, typically using algorithms that maximize the margin while minimizing classification errors.

5. Classify New Data

Use the trained SVM to make predictions.

- Steps:

1. Decision Function: For a new data point, compute which side of the hyperplane it falls on to determine its class.

6. Evaluate Model Performance

Assess how well the model performs.

Steps:

1. Metrics: Use metrics like accuracy, precision, recall, and F1 score to evaluate the model's performance on test data.

7. Tune Hyperparameters

Improve the model by adjusting parameters.

Steps:

1. Kernel Parameters: Adjust parameters specific to the chosen kernel function (e.g., degree for polynomial kernel, gamma for RBF kernel).
2. Regularization Parameter: Adjust the penalty parameter (C) that controls the trade-off between achieving a low error on the training data and minimizing the model complexity.

8. Test and Make Predictions

Apply the model to new data.

Steps:

1. Testing: Use the trained model to classify or predict outcomes for new instances.
2. Inference: Generate predictions for new data based on the decision function.

9. Deploy the Model

Implement the model in a real-world environment.

Steps:

1. Save Model: Store the model's parameters, including the support vectors and hyperplane details.

2. Integration: Deploy the model to make real-time predictions in a production environment.

CHAPTER 4

IMPLEMENTATION AND TESTING

4.1 Implementation

4.1.1. Tools Used (CASE tools, Programming languages, Database platforms)

Table 4.1 Tools Used

Category	Technology/Tool	Description
Frontend	React	JavaScript library for building user interfaces.
	CSS	Stylesheet language used to describe the look of the app.
	HTML	A standard markup language for creating web pages.
Backend	Flask (Python)	Lightweight Python web framework for building the backend.
	SQLite	A relational database management system used to store data.

4.1.2. Implementation Details of Modules (Description of procedures/functions)

There are different module descriptions. They are described below:

Sources of Data:

Eye Disease Classification Dataset:

The dataset used for eye disease prediction is ‘eye_diseases_classification’ which was extracted from Kaggle [6]

Client Module:

This module represents the front-end portion of the project, consisting of several directories. It includes a node_modules folder where the necessary npm packages are installed. Additionally, the public directory contains important files like index.js. The src folder houses all the React code, featuring the JSX components. The styles folder is dedicated to CSS files, handling the design aspects of the application. Notably, App.js serves as the main component, acting as the core of the React application.

- App.css: This file contains the CSS styles specific to the App.js component. It helps style the core component of the application.
- App.js: This is the main component of the React application, responsible for rendering the initial interface and managing the application's core logic.
- App.test.js: This file contains the test cases for the App.js component. It ensures that the component behaves as expected.
- index.css: Contains the global styles that are applied across the entire application.
- index.js: This is the entry point of the React application. It renders the App component into the DOM and may also include important configurations like ReactDOM.render() or setting up Redux.
- login.css: Holds the CSS styles specifically for the login-related components.
- login.js: This is likely the file responsible for rendering the login form or handling the login logic.
- logo.svg: A vector-based logo image file that may be used in the application header or elsewhere.
- modelAccuracy.css: Contains CSS related to styling elements that deal with displaying model accuracy or related statistics.
- modelAccuracy.js: Likely a React component file that displays or calculates the accuracy of a machine learning model.
- reportWebVitals.js: This file is used for measuring and reporting web performance metrics like loading time and user interactions.
- setupTests.js: Contains the setup for unit tests, typically configuring tools like Jest or Enzyme.
- signup.js: Handles the sign-up form or registration logic in the application.

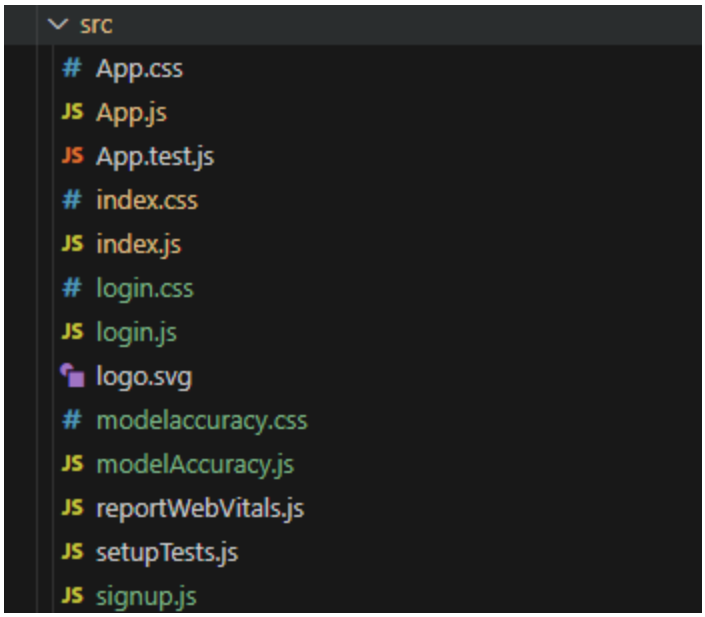


Figure 4.1 Client Module

Server Module:

This module is completely dedicated to the backend logic of the system. Whenever user, it validates and save their data to the database. It also authenticates and authorizes the user based on their roles.

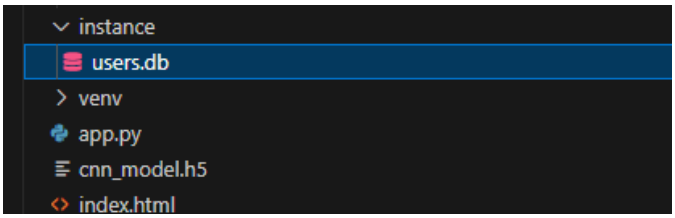


Figure 4.2 Server Module

- Venv Module: The structure shows the file organization for a Python-based machine learning project using Flask for API creation and deployment. This project appears focused on eye disease detection using various machine learning models.
- venv Directory: This directory contains the Python virtual environment, ensuring dependencies are isolated and do not conflict with other Python projects on the system.
- app.py:This Python script is the main entry point for the Flask application. It

initializes the Flask app and sets up the routes for the API, handling requests and sending responses. It may also load the machine learning models into memory when the server starts.

- models.py: This file contains the definitions and training code for the machine learning models, such as the CNN, KNN, and SVM models. These models are trained on a dataset to predict eye disease and are serialized into .h5 and .pkl files.
- cnn_model.h5, knn_model.pkl, svm_model.pkl: These files are serialized versions of the trained models. The .h5 file is typically associated with models trained using libraries like Keras/TensorFlow, whereas .pkl files are used for models trained with libraries like scikit-learn. These models are loaded by the Flask application to make predictions in real-time.
- Lib, Include, Scripts, pyvenv.cfg: These directories and files are part of the Python virtual environment setup. They contain the actual Python libraries installed, scripts for activating the environment, and configuration files.

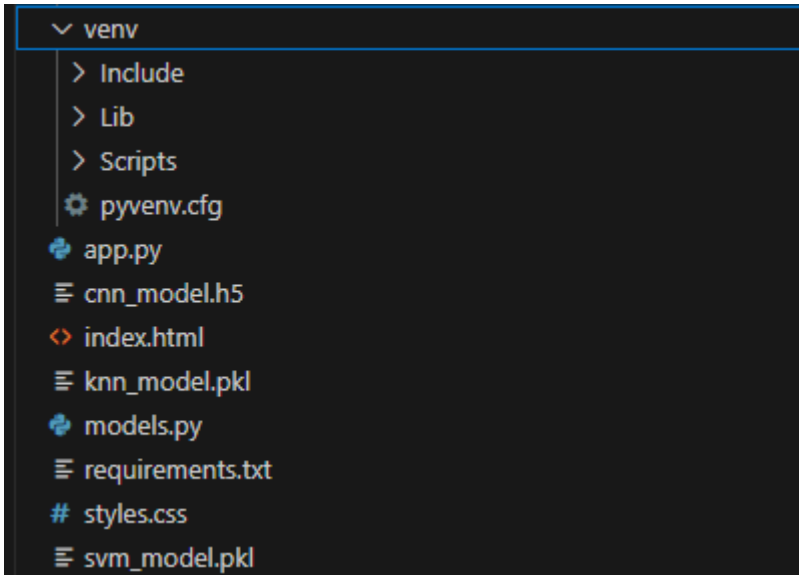


Figure 4.3 Venv Module

```python
# Function to train KNN model
def train_knn():
    start_time = time.time()
    print("Training KNN model...")
    try:
        knn_model = KNeighborsClassifier(n_neighbors=3)
        knn_model.fit(X_train_flattened, y_train)
        with open("knn_model.pkl", "wb") as f:
            pickle.dump(knn_model, f)
        elapsed_time = time.time() - start_time
        print(f"KNN model trained and saved in {elapsed_time:.2f} seconds.")
        return knn_model
    except Exception as e:
        print(f"Error training KNN model: {e}")
        exit(1)
```

Figure 4.4 Using Custom KNN Model to train and test dataset

In Figure 4.4 the train_knn function is structured to train a K-Nearest Neighbors (KNN) classifier, a type of machine learning model useful for classification tasks. The function operates by first recording the start time to calculate the duration of the training process. It prints a message indicating the initiation of the KNN model training. Within a try block, the function initializes the KNN model with n_neighbors=3, specifying that the model should consider the three nearest points in the dataset to determine the classification. The model is then trained using X_train_flattened and y_train datasets, which are typically the features and labels respectively. After training, the model is serialized using Python's pickle module and saved to a file named knn_model.pkl. This allows the model to be reused later without the need to retrain. The function calculates and prints the elapsed time since the training began, providing feedback on how long the training took. If there are any errors during the model training process, such as issues with the input data or the training itself, the function catches the exception and prints an error message. Finally, if an exception occurs, the function exists with a status code of 1, indicating an error condition. This robust error handling ensures that any issues are clearly communicated and that the function halts execution when critical failures occur.

```python
# Function to train SVM model
def train_svm():
    start_time = time.time()
    print("Training SVM model...")
    try:
        svm_model = SVC(kernel='linear')
        svm_model.fit(X_train_flattened, y_train)
        with open("svm_model.pkl", "wb") as f:
            pickle.dump(svm_model, f)
        elapsed_time = time.time() - start_time
        print(f"SVM model trained and saved in {elapsed_time:.2f} seconds.")
        return svm_model
    except Exception as e:
        print(f"Error training SVM model: {e}")
        exit(1)
```

Figure 4.5 Using SVM Model to train and test dataset

In Figure 4.5, the `train_svm` function orchestrates the training of a Support Vector Machine (SVM) classifier, renowned for its effectiveness in high-dimensional spaces. The procedure initiates by logging the start time to monitor how long the training will take. A prompt is displayed to indicate the start of the SVM training session. In the try block, an SVM model is instantiated with a 'linear' kernel to handle linear separations in the data. This model undergoes training using the datasets `X_train_flattened` for features and `y_train` for labels. Upon successful completion of the training, the model is serialized using Python's `pickle` module and saved to `svm_model.pkl`, facilitating future reuse without the need to retrain. The function then computes the elapsed time since the start of the training and displays this duration, providing a quantitative insight into the training efficiency.Should any exceptions arise during training, such as data misalignments or computational errors, the function captures these, outputs an error message detailing the issue, and terminates with a status code of 1. This error management approach ensures that any critical issues are promptly addressed, maintaining the stability and reliability of the training process.

```python
# Function to train CNN model
def train_cnn():
    start_time = time.time()
    print("Training CNN model...")
    try:
        X_train_cnn = X_train.reshape(-1, 128, 128, 1)
        X_test_cnn = X_test.reshape(-1, 128, 128, 1)
        y_train_cnn = to_categorical(y_train, num_classes=len(np.unique(y_train)))
        y_test_cnn = to_categorical(y_test, num_classes=len(np.unique(y_test)))

        cnn_model = Sequential([
            Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(128, 128, 1)),
            MaxPooling2D(pool_size=(2, 2)),
            Conv2D(64, kernel_size=(3, 3), activation='relu'),
            MaxPooling2D(pool_size=(2, 2)),
            Flatten(),
            Dense(128, activation='relu'),
            Dropout(0.5),
            Dense(len(np.unique(y_train)), activation='softmax')
        ])

        cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        cnn_model.fit(X_train_cnn, y_train_cnn, epochs=10, validation_split=0.2, batch_size=32)
        cnn_model.save('cnn_model.h5')
        elapsed_time = time.time() - start_time
        print(f"CNN model trained and saved in {elapsed_time:.2f} seconds.")
        return cnn_model
    except Exception as e:
        print(f"Error training CNN model: {e}")
        exit(1)
```

Figure 4.6 Using CNN model to train and test dataset

In figure 4.6, The `train_cnn` function is specifically tailored to train a Convolutional Neural Network (CNN), a class of deep neural networks highly effective for analyzing visual imagery. The training begins with the recording of the start time, to measure the total training duration. The function announces the start of the CNN model training through a console message.Within the try block, the function reshapes `X_train` and `X_test` into the format required by CNNs, which includes specifying the image dimensions and the number of channels (in this case, 128x128 pixels with one color channel). The target variables, `y_train` and `y_test`, are converted into categorical format using `to_categorical`, accounting for the number of unique classes present.The architecture of the CNN is defined next, featuring layers like `Conv2D` for convolution operations, `MaxPooling2D` for downsampling, and `Dense` layers for prediction. The model includes activation functions like ReLU and a softmax output layer for multi-class classification. The model also incorporates a `Dropout` layer to reduce overfitting by randomly setting a fraction of input units to 0 at each update during training.The CNN is compiled with the Adam optimizer and categorical crossentropy as the loss function. It is trained on the reshaped and categorized data over 10 epochs with a validation split of 20% and a batch size of 32. After training, the model is saved to `cnn_model.h5`.

If the training encounters any exceptions, such as issues with data formatting or memory constraints, the function captures these, outputs a detailed error message, and exits with a status code of 1. This robust error handling ensures that problems are managed effectively, keeping the training process reliable and transparent about its progress and issues.

```
# Evaluate models
print("Evaluating models...")
try:
    start_time = time.time()
    svm_accuracy = accuracy_score(y_test, svm_model.predict(X_test_flattened))
    knn_accuracy = accuracy_score(y_test, knn_model.predict(X_test_flattened))
    cnn_accuracy = cnn_model.evaluate(X_test.reshape(-1, 128, 128, 1), to_categorical(y_test), verbose=0)[1]

    elapsed_time = time.time() - start_time
    print(f"Model evaluation completed in {elapsed_time:.2f} seconds.")
    print(f"SVM Accuracy: {svm_accuracy * 100:.2f}%")
    print(f"KNN Accuracy: {knn_accuracy * 100:.2f}%")
    print(f"CNN Accuracy: {cnn_accuracy * 100:.2f}%")
except Exception as e:
    print(f"Error during evaluation: {e}")
    exit(1)
```

Figure 4.7 Code for Evaluating Models

In figure 4.7, the code evaluates the performance of three machine learning models—SVM, KNN, and CNN—on a test dataset. It begins by printing a message to indicate the start of the evaluation. The current time is recorded to measure how long the evaluation takes. For each model, the code predicts the labels for the test data and calculates the accuracy by comparing these predictions to the true labels. The CNN model requires the test data to be reshaped to fit its input format, and its accuracy is calculated using a dedicated evaluation method that returns multiple metrics. After all models are evaluated, the elapsed time is calculated and displayed along with the accuracy results, formatted as percentages. The entire process is wrapped in a try-except block to handle any potential errors, ensuring that if an exception occurs, it prints the error message and exits the program gracefully. Overall, this code efficiently summarizes the performance of the models while providing error handling for robustness.

4.2 Testing

Testing is the process of detecting errors. It performs a very critical role for quality assurance and for ensuring the reliability of software. The results of testing are used later during maintenance also.

4.2.1 Test Cases of Unit Testing

Unit Testing is a software development-process in which the smallest testable parts of an application called units are individually and independently scrutinized for proper operation. The system contains different types of individual types of individual parts that are tested. Some of the test cases are:

Table 4.2 User Registration

SN	Action	Input	Expected Outcome	Actual Outcome	Test Result
1	Launch Application	Localhost:3000/	Login Page	Login Page	Pass
2	Submit without any Details	Null	Fill the necessary field.	Fill the necessary field.	Pass
4	Enter email of incorrect format	Name: test Email: test.com Password: test123	Please include an in the email address.	Please include an in the email address. 'test.com' is missing an @	Pass
5	Enter correct details	Name: test Email: test@gmail.com Password: test123	Registration Successful	Registration Successful	Pass

Table 4.3 User Login

SN	Action	Input	Expected Outcome	Actual Outcome	Test Result
1	Launch Application	Localhost:3000/	Login Page	Login Page	Pass

2	Submit without any Details	Null	This field cannot be empty	This field cannot be empty	Pass
3	Enter incorrect email	Email: testtesttest@gmail.com Password: 123	An error occurred	An error occurred	Pass
4	Enter incorrect password	Email: test@gmail.com Password: Alish	Incorrect Password	Incorrect Password	Pass
5	Enter correct details	Email: test@gmail.com Password: test123	Redirected to Interface	Redirected to Interface	Pass

4.2.2 Test Cases for System Testing

System testing is an overall testing of the system after integrating all the functions of the project. When all the functions are integrated then system testing is done.

Table 4.4 User Interface

SN	Action	Input	Expected Outcome	Actual Outcome	Test Result
1.	Launch Application	Localhost:3000/register	Registration Page	Registration Page	Pass
2.	Register new user	First name: Test Email: test@gmail.com Password: test123	Registered successfully	Registered Successfully	Pass
3.	Login by same username	Email:test1234 Password: test123	Login Successful	Redirected to Interface	Pass
4.	Upload image without Logging in	Image.png	Redirect to login page	Redirect to login page	Pass
5.	View Accuracy without Logging in	Button Clicked	Redirect to login page	Redirect to login page	Pass

6.	View Accuracy without Logging in	Button Clicked	Please upload an image before viewing the model accuracy.	Please upload an image before viewing the model accuracy.	Pass
7.	Provide non-retinal image.	Invalidimage.png	Invalid image: Only retinal images are allowed.	Invalid image: Only retinal images are allowed.	Pass
8.	Provide valid image	100_right.png	Get predictions	Get predictions	Pass
9.	Predict Eye Disease	Cataract.png	Cataract	Cataract	Pass

4.3 Result Analysis

The system was tested using unit testing, which is a method that evaluates each part of the software to ensure it functions correctly. The tests showed that the system can effectively carry out its intended tasks. To optimize the project's impact and enhance inclusivity within the community, a comprehensive report is generated, summarizing the performance metrics of various models used in this project. Overall, the results indicate that the project has successfully met its goals and objectives. However, there is still room for growth and improvement.

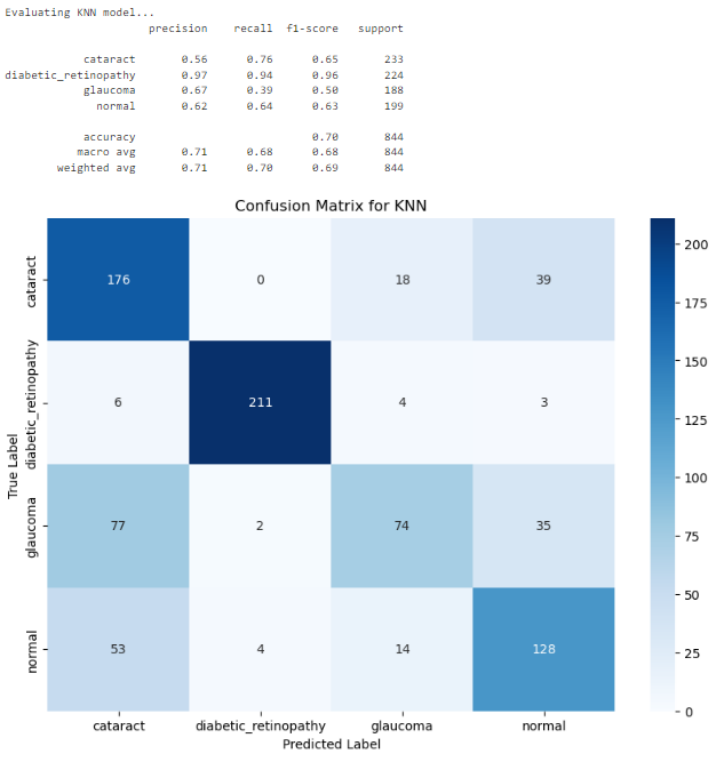


Figure 4.8 Confusion matrix for KNN Model

In Figure 4.8 the confusion matrix visualizes the performance of a KNN model on medical image classification. It shows the predicted class labels (horizontal axis) against the true class labels (vertical axis). Diagonal cells represent correct classifications (e.g., 176 cataract images correctly identified), while off-diagonal cells show misclassifications. Overall accuracy (0.71) and class-specific metrics (precision, recall, f1-score) provide a comprehensive evaluation of the model's performance. The color-coded heatmap visually highlights the distribution of correct and incorrect predictions, aiding in identifying areas for improvement.

```
Evaluating models...
Evaluating SVM model...
                    precision    recall  f1-score   support

          cataract       0.68      0.79      0.73       233
diabetic_retinopathy       0.97      1.00      0.98       224
          glaucoma       0.67      0.53      0.59       188
            normal       0.74      0.72      0.73       199

          accuracy                           0.77       844
         macro avg       0.76      0.76      0.76       844
      weighted avg       0.77      0.77      0.77       844
```

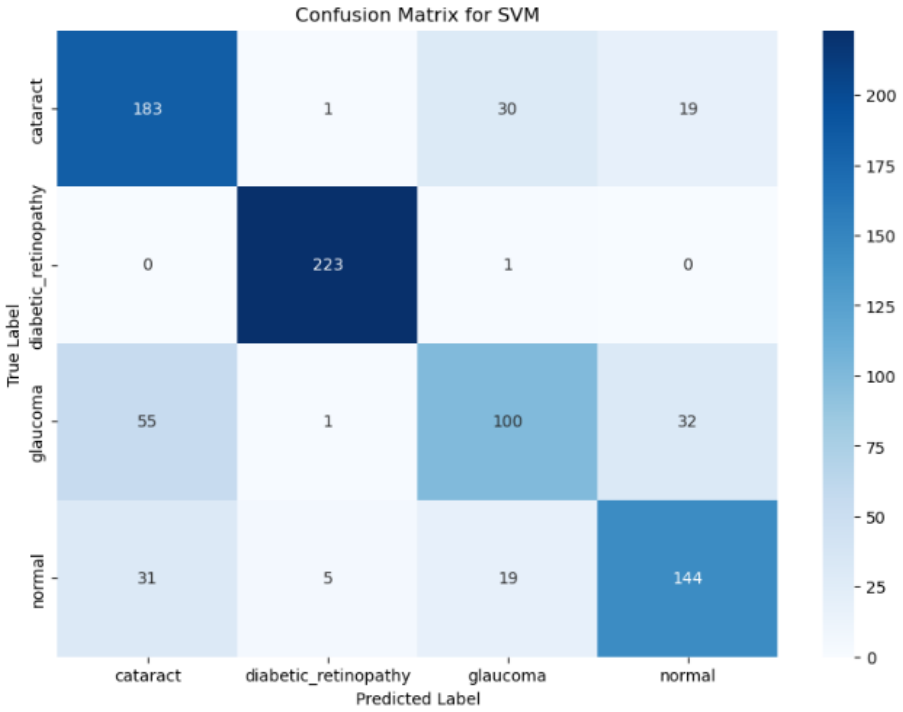


Figure 4.9 Confusion matrix for SVM

In Figure 4.9 The confusion matrix visualizes the performance of an SVM model on medical image classification. It shows predicted class labels against true labels, with diagonal cells representing correct classifications (e.g., 183 cataract images correctly identified) and off-diagonal cells indicating misclassifications. Overall accuracy (0.77) and class-specific metrics (precision, recall, f1-score) evaluate the model's performance. The color-coded heatmap visually highlights the distribution of correct and incorrect predictions, aiding in identifying areas for improvement.

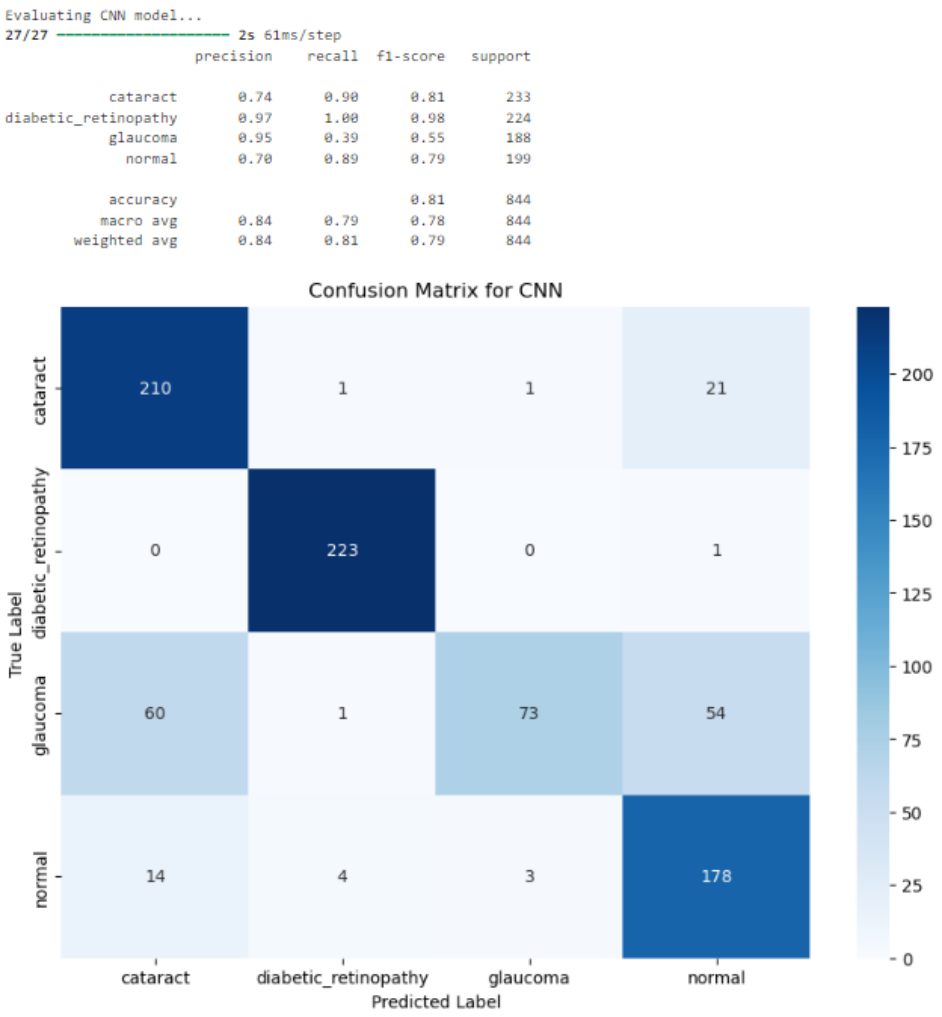


Figure 4.10 Confusion matrix for CNN Model

In Figure 4.10 the confusion matrix visualizes the performance of a CNN model on medical image classification. It shows predicted class labels against true labels, with diagonal cells representing correct classifications (e.g., 210 cataract images correctly identified) and off-diagonal cells indicating misclassifications. Overall accuracy (0.84) and class-specific metrics (precision, recall, f1-score) evaluate the model's performance. The color-coded heatmap highlights the distribution of correct and incorrect predictions, aiding in identifying areas for improvement.

CHAPTER 5

CONCLUSION AND FUTURE RECOMMENDATION

5.1 Conclusion

This project, “Eye Disease Detection System,” is a web-based application designed to provide users with a convenient and accessible platform for predicting eye diseases based on their input. The system allows users to enter specific symptoms and personal health information(images), and, based on this data, it generates predictions about potential eye diseases. If a potential eye disease is detected, the application facilitates scheduling appointments with qualified eye specialists, ensuring users receive timely and appropriate care. Despite its innovative approach, the current accuracy levels of the system have been found to be average. This underscores the importance of ongoing development and refinement to enhance its predictive capabilities. To address this issue and improve the reliability of the predictions, it is crucial to incorporate additional algorithms into the system. By expanding the range of algorithms used, the application can analyze user data more comprehensively, ultimately leading to more accurate and dependable predictions.

5.2 Lesson Learnt/ Outcome

The development of the "Eye Disease Detection System" provided valuable learning outcomes across various domains. It enhanced the author's understanding of web application development using technologies such as React.js and Python, while also emphasizing the significance of data analysis and machine learning in healthcare applications. The project highlighted the importance of user input for accurate predictions and the challenges associated with achieving reliability and accuracy in predictive systems. Additionally, it fostered essential project management skills, reinforcing the need for effective planning and execution. This experience cultivated a commitment to continuous learning, motivating the author to seek further knowledge and improvement, ultimately demonstrating the real-world impact of technology in enhancing patient care through early detection and timely consultations with specialists.

5.3 Future Recommendation

For future enhancements of the "Eye Disease Detection System," several recommendations can be considered. First, incorporating advanced machine learning algorithms, such as deep learning models, could significantly improve the accuracy of predictions. Exploring ensemble methods that combine multiple algorithms may also enhance reliability. Second, conducting extensive testing with diverse datasets can help fine-tune the model and address potential biases in predictions. Additionally, implementing a user feedback mechanism would allow for continuous improvement based on real-world user experiences. Furthermore, enhancing the user interface to make it more intuitive and user-friendly can increase engagement and accessibility for a broader audience. Integrating features such as telemedicine options for virtual consultations with eye specialists could streamline the process and improve user convenience. Lastly, ongoing training and education in emerging technologies will empower the development team to stay current with industry advancements, ensuring the system remains competitive and effective in delivering reliable eye health predictions.

REFERENCES

[1] A. Patel, B. Kumar, and C. Singh, "Automated Detection of Diabetic Retinopathy Using Convolutional Neural Networks," Journal of Medical Imaging and Health Informatics, Vol. 13, 2023.

[2] D. Sharma and E. Jain, "Glaucoma Detection with Deep Learning Techniques," International Journal of Ophthalmology Research, 2021.

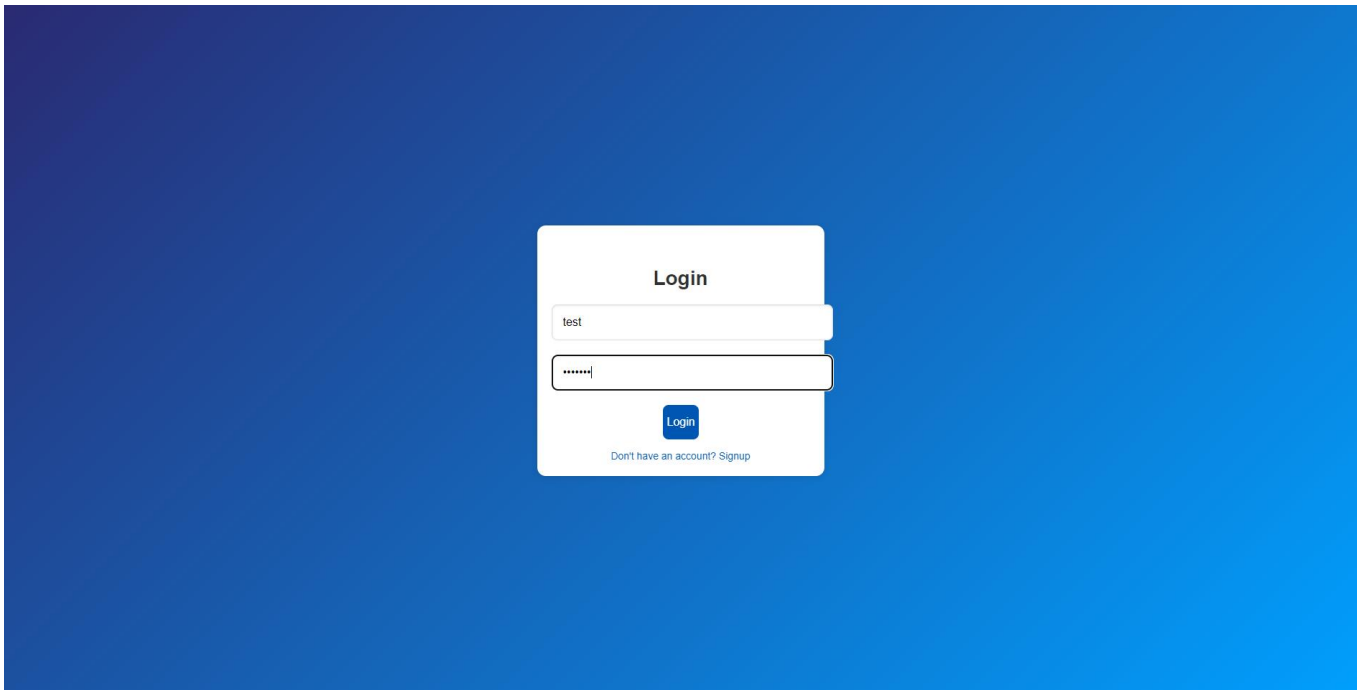
[3] F. Lee and G. Zhou, "Cataract Classification and Detection Using Deep Convolutional Neural Networks," Journal of Vision and Eye Research, 2023.

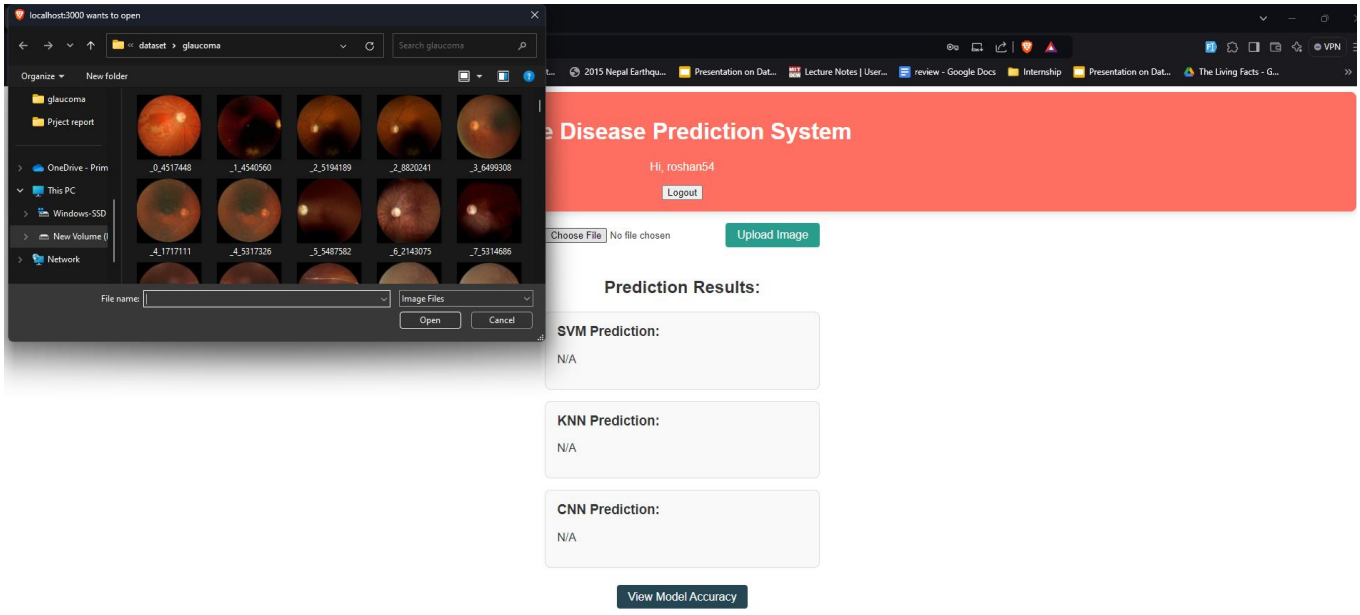
[4] H. Patel, I. Kumar, and J. Singh, "Comparative Study of CNN and SVM for Eye Disease Classification," Computational Vision and Image Understanding, 2023.

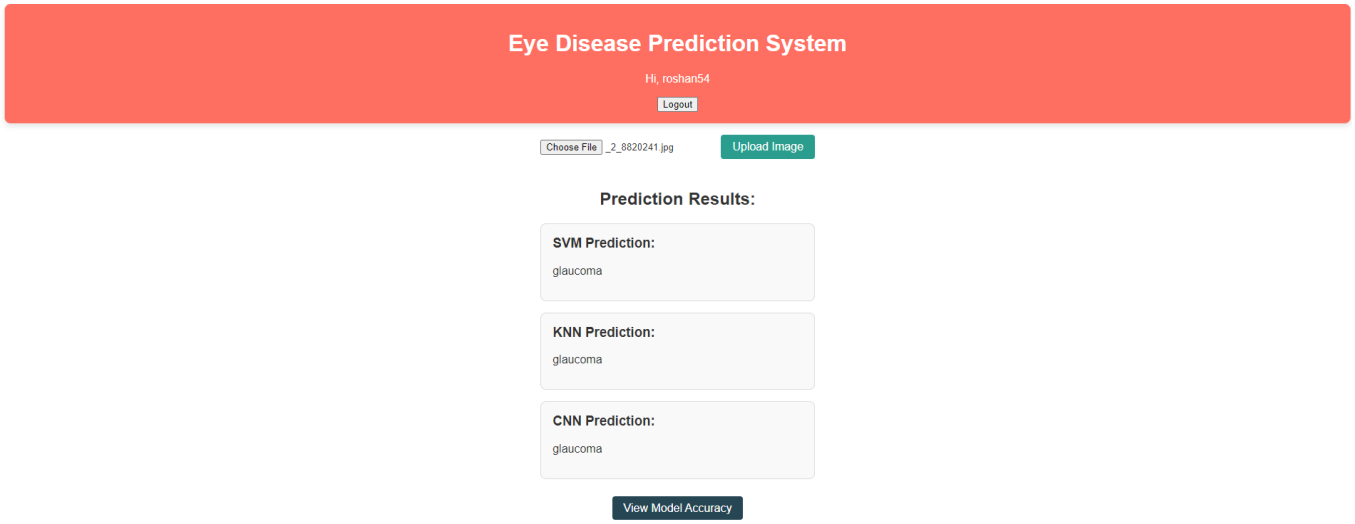
[5] K. Brown and L. Smith, "Integrating Machine Learning Models for Comprehensive Eye Disease Detection," Artificial Intelligence in Healthcare, 2020.

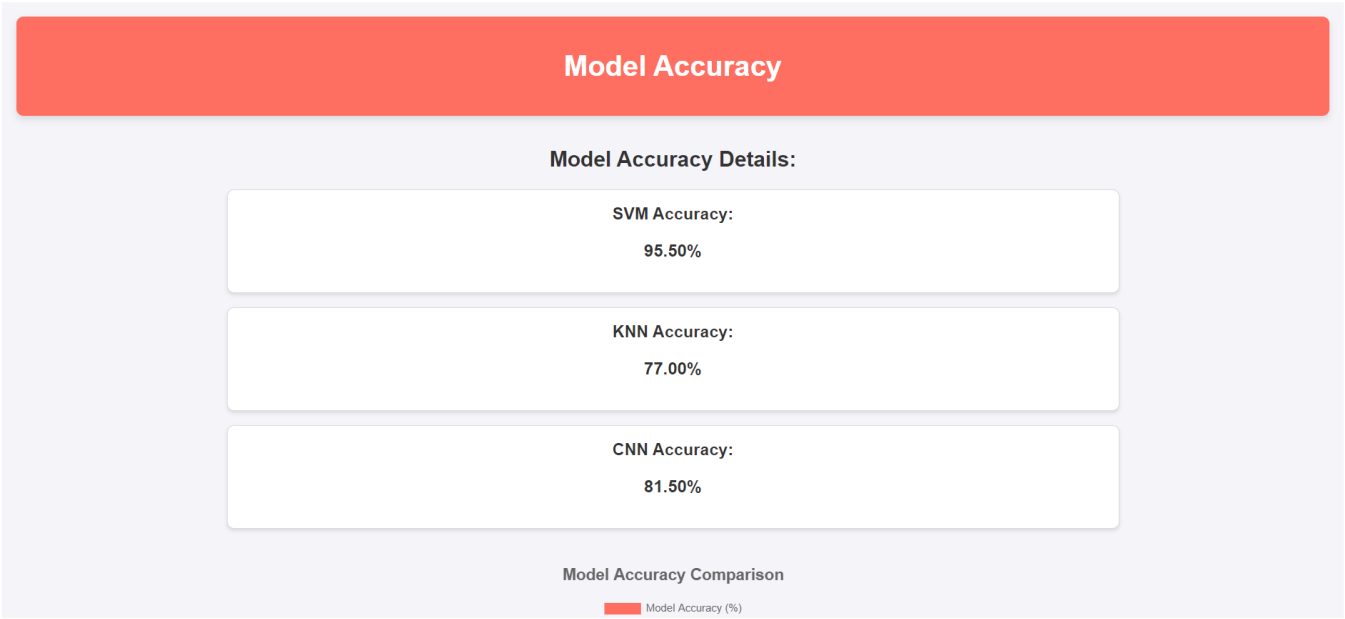
[6] "eye_diseases_classification", www.kaggle.com. https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification (accessed Jun 30,2023)

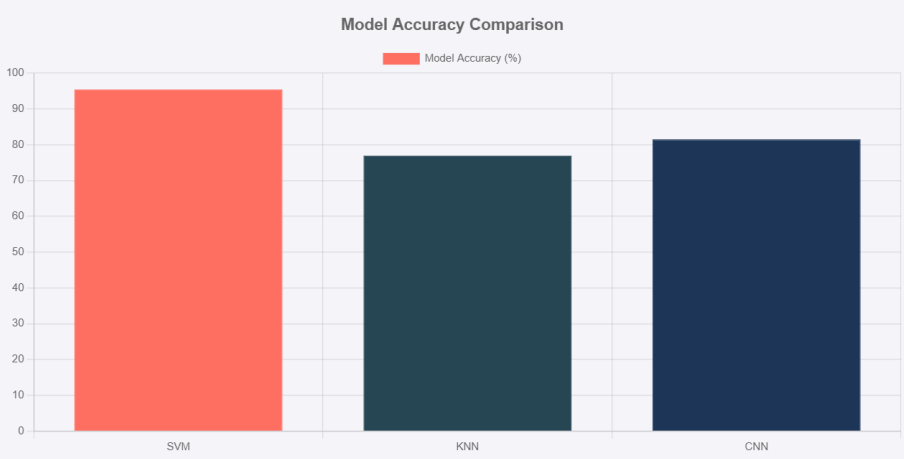
APPENDICES











Back to Home