InSight: Automated Detection of Cataracts in Retinal Image Data using a Convolutional Neural Network

Njuguna Faith Nyambura

150325

ICS 4 D

Supervisor Name

Dr. Kennedy Ronoh

An Informatics and Computer Science Project Documentation Submitted to the School of Computing and Engineering Sciences in partial fulfillment of the requirements for the award of a Degree in Bachelor of Science in Informatics and Computer Science

School of Computing and Engineering Sciences

Strathmore University

Nairobi, Kenya

November 2025

Declaration and Approval

I declare that this work has not been submitted to Strathmore University or any other University for the award of a Degree in Bachelor of Science in Informatics and Computer Science or any other Degree. To the best of my knowledge and belief, the research documentation contains no material previously published or written by another person except where due reference is made in the research documentation itself.

Student Name: **Njuguna Faith Nyambura**

Student Admission Number: **150325**

Sign: [illegible]

Date: 23/11/2025

Supervisor's Name: **Dr. Kennedy Ronoh**

Sign: K.Ronoh

Digitally signed by K.Ronoh
DN: OU=SCES, O=Strathmore University, CN=K.Ronoh, E=kronoh@strathmore.edu
Reason: I am approving this document
Location: Nairobi
Date: 2025.11.24 13:18:27+03'00'
Foxit PDF Reader Version: 2025.2.0

Date:

Acknowledgement

I would like to begin by thanking God for granting me the clarity, strength and perseverance to undertake and progress through this project. I truly appreciate my supervisor, Dr. Kennedy Ronoh, for his constructive feedback and invaluable guidance throughout this journey. I also extend my gratitude to Strathmore University for incorporating this project into the curriculum as it has greatly enhanced my research skills and broadened my understanding.

Abstract

Cataracts remain a leading cause of preventable blindness worldwide, particularly in low-resource settings where diagnostic delays worsen outcomes. However, existing automated diagnostic tools often lack seamless integration into clinical workflows, suffer from ‘black box’ opacity that hinders clinician trust or require computational resources unavailable in primary care facilities. To better support frontline health workers, this project presents InSight, a clinician-focused web application designed for use by nurses and doctors to speed up screening and streamline triage.

InSight utilizes a Convolutional Neural Network (CNN) to analyze uploaded ocular fundus photographs and return an interpretable diagnostic prediction. The system prioritizes a secure, multi-user workflow, fast inference times, and visual explanations using Grad-CAM heatmaps to increase clinician trust and facilitate faster patient referral.

An experimental methodology was employed, starting with acquiring and processing a large-scale fundus image dataset (ODIR). The images were preprocessed and split into training and validation sets. The CNN model (ResNet18) was subsequently trained using PyTorch, employing a Focal Loss function to effectively manage the dataset's ingrained class imbalance. The trained model was integrated into a FastAPI backend to handle predictions and Grad-CAM generation. The frontend was built with Streamlit, providing a secure, role-based interface for a nurse or a doctor, with database integration through Supabase for tracking patient screening history.

Performance evaluation demonstrated that the model achieved a validation accuracy of 99.52% and an exceptionally low validation loss of 0.0006. These results confirm the system's capability to distinguish cataracts from healthy fundus images with a high degree of reliability. InSight provides an efficient screening aid that speeds up patient assessment, supports early detection and helps reduce avoidable vision loss in underserved populations.

Keywords: Cataract Detection, Fundus Photography, Convolutional Neural Network, PyTorch, Grad-CAM, FastAPI, Streamlit

Table of Contents

List of Figures

List of Tables

List of Abbreviations

API	– Application Programming Interface
AUC	– Area Under the Curve
CNN	– Convolutional Neural Network
FN	– False Negative
FNR	– False Negative Rate
FP	– False Positive
IOL	– Intraocular Lens
MLP	– Multi-Layer Perceptron
ODIR	– Ocular Disease Intelligent Recognition
P	– Precision
ROC	– Receiver Operating Characteristic
TPR	– True Positive Rate

Chapter 1: Introduction

1.1 Background

Cataracts are the leading cause of blindness worldwide, contributing to nearly 45% of all cases of visual impairment, with over 94 million people affected globally (Vision Loss Expert Group, 2023). This condition arises when the eye's natural lens becomes cloudy, typically because of aging, leading to blurred or dim vision. In more advanced stages, cataracts can cause total blindness if not treated surgically. Despite the availability of effective treatment, the prevalence of blindness caused by cataracts remains very high in low and middle-income countries, including Kenya, where cataracts are responsible for over 40% of blindness cases (Gichangi et al., 2022).

While the definitive diagnosis of cataracts is performed with a slit-lamp biomicroscope, mass screening for various eye diseases increasingly relies on ocular fundus photography. A fundus camera captures a digital image of the retina, optic nerve and vasculature at the back of the eye. This single image can be used to screen for multiple conditions, including diabetic retinopathy, glaucoma, and macular degeneration. Advanced cataracts are also visible on fundus images as they obscure the view of the retina, making these images a viable tool for cataract screening, especially in high-volume, low-resource settings.

The primary challenge, however, remains the significant shortage of trained ophthalmologists required to interpret these images. This creates a critical bottleneck, where images are captured but patients face long delays in receiving a diagnosis (Gichangi et al., 2022).

This project, InSight, addressed this gap by creating a lightweight web-based screening application tailored for use by nurses and doctors. The system enables clinicians to upload ocular fundus photographs for quick analysis by a convolutional neural network (CNN). Predictions are accompanied by interpretable visual explanations through Grad-CAM heatmaps, assisting in decision-making and supporting timely patient referral. By prioritizing ease of use, speed and deployment flexibility, InSight strengthens frontline cataract screening services, reduces diagnostic delays and improves efficiency in busy healthcare environments.

1.2 Problem Statement

While fundus cameras are becoming more common in mass screening camps, the critical shortage of trained specialists available to interpret the thousands of images generated remains the primary bottleneck. This diagnostic gap creates significant operational backlogs, leading to prolonged patient waiting times, increased travel burdens for rural populations seeking care and diagnostic delays that allow preventable conditions to progress (Tham et al., 2022). Consequently, many patients are only diagnosed after significant vision loss has already occurred, which can reduce the overall effectiveness and positive impact of surgical interventions. While artificial intelligence has demonstrated strong performance in research settings for diagnosing other eye diseases, there remained a noticeable absence of reliable, AI-powered systems integrated into a simple clinical workflow for early cataract screening using fundus images. Many existing AI models functioned as black boxes, providing a prediction with no explanation, which limited clinician trust and adoption. Few models had been successfully packaged into a user-friendly, end-to-end tool that could support frontline health workers, provide interpretable results and connect to a persistent patient record system.

This project addressed these specific challenges. The core problem identified was not just the need for an accurate algorithm, but the absence of a complete, secure and interpretable system built for clinical use. There was a clear need for a solution that provided rapid, interpretable predictions (using Grad-CAM), stored results in a secure database and featured role-based access for different clinical staff (nurses and doctors). InSight was therefore developed to fill this gap, providing a tool to streamline the entire screening, referral and data-tracking process.

1.3 General Objective

To develop a deep learning-based system that automatically detects cataracts from ocular fundus images.

1.3.1 Specific Objectives

i. To review the current methods used to diagnose cataracts and the challenges associated with these approaches in clinical settings.
ii. To analyze and prepare a suitable dataset of eye images for training and evaluating a convolutional neural network (CNN) model.

iii. To design a clinician-focused web interface to support nurses and doctors.
iv. To develop a deep learning-based system that automatically detects cataracts from ocular fundus images.
v. To test the system for accuracy, reliability and usability by evaluating its performance on unseen image data and through user feedback from healthcare professionals.

1.3.2 Research Questions

i. What are the current methods used to diagnose cataracts and what challenges do these methods face in clinical settings?
ii. How can a high-quality and diverse eye image dataset be collected and prepared to support CNN-based cataract detection?
iii. What design strategies are effective in building a clinician-focused web interface suitable for use by nurses and doctors?
iv. How can a deep learning system be developed to accurately detect cataracts from ocular fundus images?
v. How effective, accurate and user-friendly is the developed CNN-based cataract detection system when tested with real-world data and evaluated by healthcare professionals?

1.4 Justification

The developed system was utilized to support the automation of the cataract screening process using ocular fundus photographs. As a result, it provided a direct technological solution to the critical diagnostic delays caused by the shortage of trained ophthalmologists. The system was designed to cut the amount of time required for a frontline health worker, such as a nurse or general doctor, to make an effective triage decision, thereby helping to reduce errors caused by high patient volumes or exhaustion. The developed model, which achieved 99.5% validation accuracy, also provided a consistent and highly accurate interpretation baseline. This addressed the challenge of diagnostic variability that can arise from a clinician's individual experience level with fundus images. This was possible because the system automatically interpreted the image, provided a clear binary prediction (Cataract Detected or No Cataract Detected) and offered an immediate clinical recommendation (Urgent or Routine).

Furthermore, by integrating this model into a full-stack application with role-based access, a patient history log and an analytics dashboard, the system provided a complete and reliable workflow. This addressed a common gap in academic models, which often lack the features for secure data management in a clinical setting. By aligning with global efforts to reduce avoidable blindness through technology-driven support, the system represented not only a technical innovation but also a socially responsible and ethically grounded solution (Shimizu et al., 2023).

1.5 Scope and Limitations

1.5.1 Scope of the Project

This project focused on developing a CNN-based cataract screening system designed to analyze ocular fundus photographs from the ODIR dataset. It was built for use by healthcare professionals, such as nurses and doctors, in clinical settings. The system was built with a Streamlit frontend and a PyTorch and FastAPI backend. It incorporated image preprocessing, binary cataract detection and visual explanations through Grad-CAM heatmaps. The system also featured a secure, role-based login and a Supabase database for persisting screening results and enabling patient history lookups.

1.5.2 Limitations in the Project

The model's performance was dependent on the quality and labeling of the ODIR dataset, which may not fully represent all populations or camera types. Image clarity and artifacts such as dust or poor lighting impacted prediction accuracy. In addition, the system was intended as a screening aid and not a substitute for professional diagnosis, which limited its clinical applicability in certain regulatory environments.

1.5.3 Delimitations in the Project

To ensure a focused and achievable project scope, several key boundaries were deliberately established. The system's primary function was limited to binary classification, distinguishing between Cataract and No Cataract based on the image provided. Consequently, the model was not developed to perform more complex tasks such as grading the severity of the cataract. This functional boundary reinforced the system's intended role as a screening support tool to aid clinicians, not to act as a standalone automated diagnostic device. The final clinical diagnosis and subsequent treatment plan properly remained the exclusive responsibility of the qualified healthcare professional. From a technical integration perspective, the project's scope did not include the complex task of direct integration with hospital Electronic Health Record (EHR) systems. This was delimited due to the significant additional challenges related to data interoperability standards, API security and patient data privacy regulations. The model's training was highly specialized as it was focused only on identifying the visual signatures of cataracts. Therefore, the detection or diagnosis of any other ocular pathologies, such as diabetic retinopathy or glaucoma, was explicitly outside the project's boundaries. In addition, this was a software-only solution. The project did not include the research or development of any physical hardware for image capture, as it was designed to operate using images from existing fundus cameras.

Chapter 2: Literature Review

2.1 Introduction

This chapter conducts a review of existing literature regarding the chosen area of study. It covers the detection and diagnosis of cataracts using ocular fundus images, as well as work relating to existing AI models and their shortcomings. By presenting a conceptual framework, it also addresses how InSight was designed and developed.

2.2 Current Methods of Cataract Detection and their Challenges

Cataracts are the leading cause of preventable blindness worldwide, contributing to nearly 45% of all cases of visual impairment. This condition, which describes a progressive clouding of the eye's natural lens, affects over 94 million people globally (Vision Loss Expert Group, 2023). The impact is most severe in low and middle-income countries; in Kenya, for example, cataracts were responsible for over 40% of all blindness cases (Gichangi et al., 2022).

The condition arises when proteins in the lens, typically due to aging, begin to break down and clump together. As illustrated in Figure 2.1, this opacity scatters light, preventing it from focusing clearly on the retina, which leads to blurred vision and eventual blindness if left untreated.

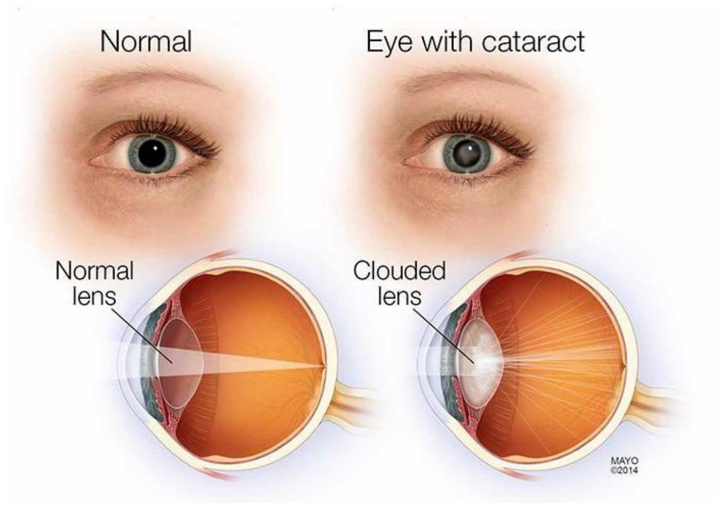


Figure 2.1: Comparison of light passing through a clear lens versus scattering through a lens with a cataract, (Mayo, 2014).

While the primary risk factor for cataracts is advanced age, other factors such as diabetes, smoking and prolonged UV exposure can accelerate their formation. The progressive nature of the condition means that with early and accurate detection, the resulting blindness is entirely preventable through a common surgical procedure (Tham et al., 2022).

The challenge addressed by this project was not the treatment of cataracts, but their diagnosis. The clinical gold standard for detection relied heavily on a slit-lamp biomicroscope examination (Mwangi et al., 2024). However, this method presented significant accessibility challenges, particularly in low-resource settings. The equipment was costly, required stable electricity and demanded specialized training to operate effectively. In Kenya, where the ophthalmologist-to-patient ratio remained critically low, these limitations created substantial barriers to timely diagnosis (Okeke et al., 2023).

Alternative screening approaches, such as mobile eye clinics, improved access but often lacked the sustainability for continuous service (Gadde et al., 2024). This diagnostic landscape underscored a pressing need for innovative, scalable solutions such as AI-assisted fundus photography analysis that could overcome the barriers of cost and specialist dependence.

2.3 Existing AI Solutions for Cataract Detection

The use of AI for cataract detection, particularly from fundus images, has been an active area of research. This section reviews three key studies that informed the development of InSight.

2.3.1 Automated Cataract Detection Using Fundus Images

A research study was carried out in 2018 to develop an automated system for identifying cataracts using retinal fundus images (Raghavendra et al., 2018). Their goal was to create a reliable screening tool to aid ophthalmologists. Their dataset contained 1,000 fundus images, which were collected from various sources and pre-processed. The methodology involved feeding these pre-processed images into pre-trained Convolutional Neural Networks (CNNs), specifically exploring architectures like AlexNet and VGG-16 as shown in Figure 2.2. These models were fine-tuned to perform a binary classification task: labeling an image as either 'Cataract' or 'Normal'.

Evaluation of the models was performed using a confusion matrix to calculate accuracy, sensitivity and specificity. The results were highly promising, with the VGG-16 model achieving an accuracy of 93.4%. This study was significant as it clearly demonstrated the feasibility of using deep learning on fundus images for automated cataract screening.

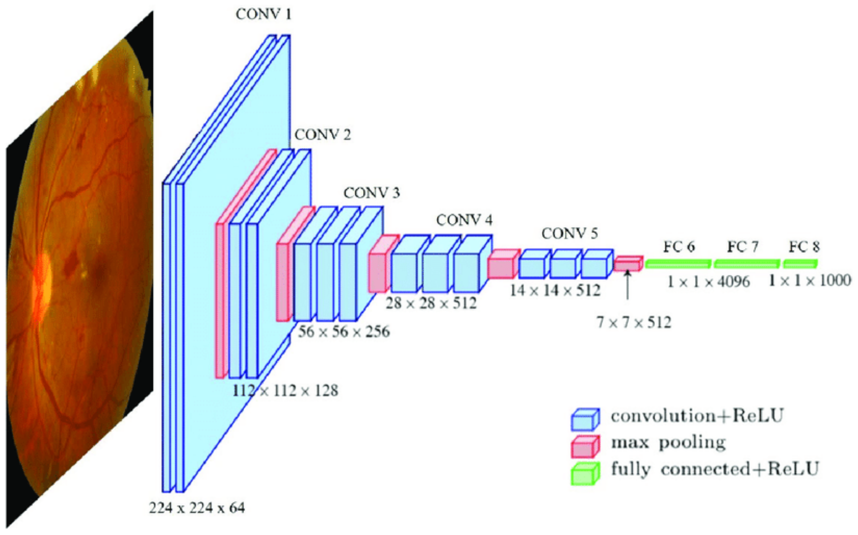


Figure 2.2: VGG-16 architecture diagram

2.3.2 Multi-Class Ocular Disease Recognition

Other research, such as the work that produced the Ocular Disease Intelligent Recognition (ODIR) dataset used in this project, focused on a more complex, multi-class problem (Li et al., 2020; ODIR-5K, 2020). The goal was not just to detect cataracts, but to build a model capable of identifying eight different ocular pathologies from a single fundus image. The dataset itself was a major contribution, consisting of 10,000 fundus images from 5,000 patients, with labels for Normal (N), Diabetes (D), Glaucoma (G), Cataract (C), Age-related Macular Degeneration (A), Hypertension (H) and more, examples of which are displayed in Figure 2.3.

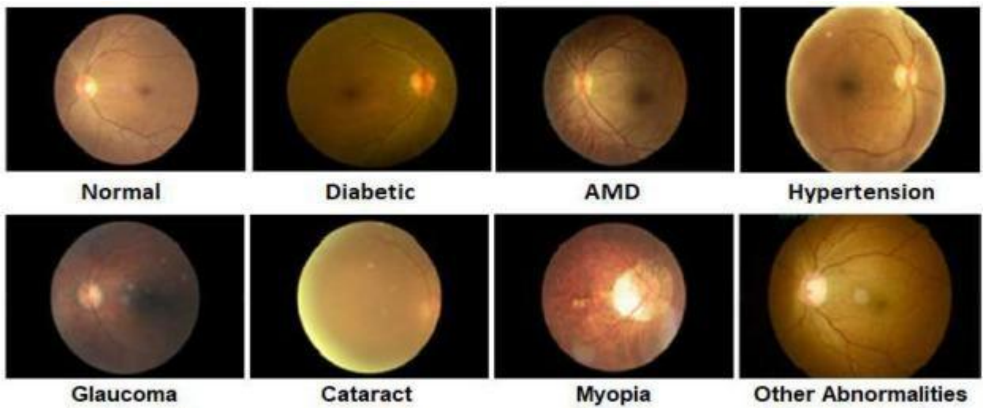


Figure 2.3: Example Images from the ODIR-5K Dataset

Models developed on this dataset, often using complex ensembles of ResNet or EfficientNet, were trained for multi-label classification. The results showed a very high Area Under the Curve (AUC) for cataract detection, proving that the visual features of a cataract were highly discernible in these images. These studies validated the dataset as a robust tool for training a cataract detection model.

2.3.3 A CNN-LSTM Combination Network for Cataract Detection

One other study also used the ODIR dataset but proposed a novel hybrid architecture (Padalia et al., 2022). Their goal was to create a low-cost diagnostic system that could classify normal and cataractous cases from fundus images. The methodology involved a CNN-LSTM-based model, conceptually depicted in Figure 2.4. The CNN (a pre-trained model like VGG-16) was used as a feature extractor to analyze the spatial information in the fundus image. These extracted features were then fed into a Long Short-Term Memory (LSTM) network, a type of model typically used for sequential data such as text or time series. The rationale was to capture complex patterns and dependencies within the image features that a standard classifier might miss.

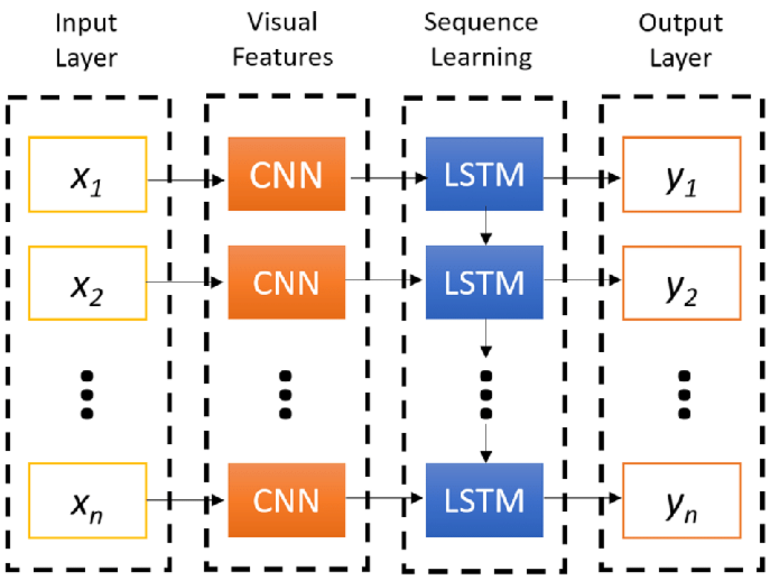


Figure 2.4: CNN LSTM architecture diagram

This hybrid model was then trained on the ODIR dataset for the binary task of cataract versus normal. The results demonstrated a competitive classification performance, suggesting that alternative architectures beyond simple CNNs could be viable for this task.

2.4 Gaps in Existing Solutions

The models trained on the full ODIR dataset, while comprehensive, were often highly complex and computationally expensive. They were typically designed to detect eight different diseases simultaneously, which was unnecessary for a clinical workflow focused solely on cataract screening. This multi-class complexity not only increased the training burden but also lengthened inference times. However, the primary gap, similar to other research, was the lack of essential application-level features. These projects did not result in a deployable tool equipped with secure authentication, patient history tracking, or a simple interface suitable for rapid screening in a clinical setting.

Other novel approaches, such as the CNN-LSTM hybrid model proposed by Padalia et al. (2022), introduced unnecessary architectural complexity. Using an LSTM, which is designed for sequential data, on a static fundus image represented a non-standard method. This made the model more difficult to interpret and more computationally expensive than a standard, proven CNN architecture such as ResNet, without offering a clear, demonstrable improvement in accuracy for this specific binary task. Most importantly, this project also remained a purely academic model, lacking the full stack required for a real-world clinical application.

InSight was designed specifically to address these gaps. It leveraged the high-quality ODIR dataset but focused only on the binary Cataract vs. No Cataract task, which solved the overkill problem and improved efficiency. Using a standard and well-understood CNN (ResNet18), the project's crucial contribution was the integration of this accurate model into a full-stack application. This was achieved using a FastAPI backend and a Streamlit frontend, which provided all the missing clinical features, including secure authentication, role-based access, a Supabase database for patient records, a patient history lookup page, an analytics dashboard and a working, integrated Grad-CAM for explainability.

2.5 Conceptual Framework

The conceptual framework for InSight, illustrated in Figure 2.5, defined the system's structured, multi-component architecture. It was designed to transform a raw ocular fundus image into an actionable and persistent clinical insight, following a clear flow from data input to final output.

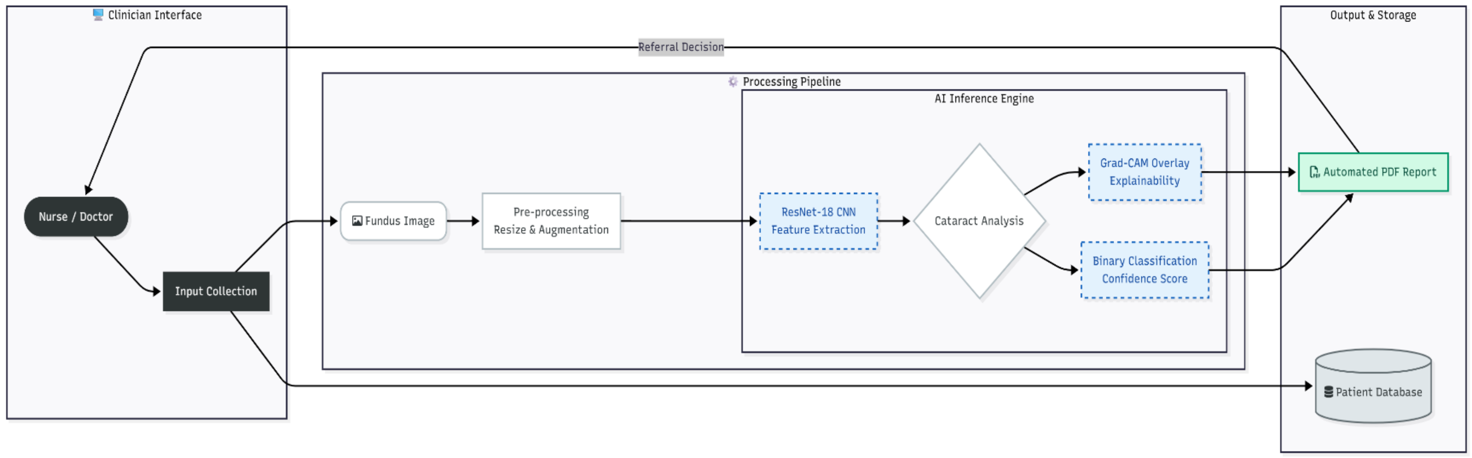


Figure 2.5: Conceptual Framework of InSight

The process began on the Streamlit frontend, which served as the user interface. A registered healthcare professional, such as a nurse or doctor, would first log in using their credentials, which were securely validated by the Supabase Authentication service. Once authenticated, the user's role was fetched from the user_roles table to customize the interface and they provided the two primary inputs: a unique Patient Identifier and the ocular fundus photograph for screening.

Upon clicking Analyze, the frontend application sent the uploaded image to the dedicated FastAPI backend server. This backend was responsible for all heavy computation. First, it preprocessed the raw image bytes, applying the necessary PyTorch transforms such as resizing and normalization, to prepare the image for the model. This processed image tensor was then fed into the trained PyTorch ResNet18 model (fundus_pytorch_model.pt) for inference, which produced a single raw logit representing the prediction.

Simultaneously, the backend generated the Grad-CAM visualization. Using PyTorch's ‘hooks’ feature, it captured the gradients and activations from the ResNet18 model's final convolutional layer (layer4[-1].conv2). These were combined to produce a heatmap, which was then overlaid onto the original image to visually highlight the features the model used for its decision. The backend then sent a JSON response back to the Streamlit frontend, containing the final prediction probability and the complete Grad-CAM image (encoded as a base64 string).

Finally, the Streamlit frontend received this response and rendered the Outputs for the clinician. This included a clear binary prediction (Cataract Detected or No Cataract Detected) based on the probability, the model's confidence score, a clinical recommendation (Urgent or Routine) and the Grad-CAM heatmap for interpretation. If the clinician chose to save the record, the frontend then wrote these screening details to the screenings table in the Supabase database, creating a persistent record that could be accessed later via the Patient History and Analytics pages.

Chapter 3: Development Methodology

3.1 Introduction

This chapter details the development methodology employed to create InSight, a web-based cataract detection system that integrates computer vision techniques with a user-friendly web interface. The chapter is structured into four main sections: the research paradigm, the software development methodology, system analysis and design, and tools and techniques. Each section provides a comprehensive narrative of the processes, models, tools and techniques that guided the development of the system. This chosen strategy ensured that both the machine learning model and the web interface were developed using a structured, iterative, and user-centered approach.

3.2 Research Paradigm

The development of InSight adopted an experimental research paradigm, which was well-suited for machine learning-based projects. In experimental machine learning research, the goal is to explore the performance of different algorithms on labeled datasets, optimize their parameters, and evaluate their predictive capability on unseen data. According to Goodfellow et al. (2016), experimental ML involves iterative testing of hypotheses using training, validation and testing datasets, allowing for verifiable optimization of models. In this project, a convolutional neural network (CNN) architecture was experimentally trained and fine-tuned on labeled eye images to accurately detect the presence of cataracts. This trained model was later integrated into the backend of the system to provide real-time cataract detection through the user-facing web interface. Model development included the following steps:

3.2.1 Data Acquisition

Data acquisition refers to the process of collecting and sourcing relevant datasets that were used to train and evaluate the machine learning model. The effectiveness of any supervised machine learning system depends significantly on the quality and diversity of the dataset used.

For this project, the Ocular Disease Intelligent Recognition (ODIR) dataset was utilized, which was obtained from the ODIR-2019 Grand Challenge (Peking University, 2019). This comprehensive dataset contains images from 5,000 patients, each providing left and right eye fundus photographs with diagnostic labels across eight categories, including cataract. Although only a portion of the dataset corresponded to cataract cases, its large scale and variety provided a robust foundation for training and evaluating the CNN model. The dataset offered both the sample size and the real-world diversity required to build a clinically relevant detection model.

3.2.2 Image Preprocessing

Image preprocessing was essential for preparing the raw retinal images from the ODIR dataset for model training. Since this dataset contained images with variations in resolution, lighting and orientation, preprocessing ensured consistency and improved model interpretability.

The images were resized to a standard input dimension (224×224 pixels) and normalized to a [0,1] pixel intensity range. Histogram equalization was also applied to enhance contrast, making cataract features more distinguishable. To further improve the model's robustness, data augmentation techniques such as horizontal flipping, slight rotations and brightness adjustments were introduced during training. These steps collectively ensured that the CNN received high-quality, standardized input that supported effective learning and generalization.

3.2.3 Model Training

Model training is the process of feeding preprocessed data into a machine learning algorithm to allow it to learn patterns and make accurate predictions. The training of the convolutional neural network was implemented using ResNet-18, a well-established architecture known for its strong balance of performance and computational efficiency.

The network was designed to perform binary classification, identifying images as either 'Cataract Detected' or 'No Cataract Detected' to align with the application's screening goal. The training dataset was split into training and validation sets in an 80:20 ratio, ensuring that performance was continuously evaluated on unseen data. Model parameters were optimized using backpropagation and the Adam optimizer. To improve interpretability, Gradient-weighted Class Activation Mapping (Grad-CAM) was integrated into the workflow, producing heatmaps that highlighted the specific regions of the eye image most influential in the model's decision. This explainability step was especially important in a clinical context, as it helps build trust in the system's predictions.

3.2.4 Model Validation and Testing

Model validation and testing assessed the performance and generalizability of the trained model using unseen data to ensure reliability and reduce overfitting. To assess the model's effectiveness, a combination of evaluation metrics was used.

Accuracy provided a general measure of performance, but more medically relevant metrics such as sensitivity (true positive rate) and specificity (true negative rate) were emphasized. These metrics are critical in medical diagnosis, where false negatives could delay necessary treatment. In addition, the area under the receiver operating characteristic (ROC) curve (AUC) was computed to understand the model's performance across various thresholds. A confusion matrix was also generated to visualize the distribution of correct and incorrect predictions. These metrics helped ensure the final model was not only technically sound but also clinically valuable.

3.3 Software Development Methodology

InSight was developed using the iterative development methodology. This approach emphasized frequent refinement of functional modules based on feedback and testing, which aligned well with the experimental nature of the machine learning model and the evolving requirements of user interaction in a healthcare app. According to Pressman (2010), iterative methodologies allow for flexibility and continuous improvement, which is particularly beneficial in projects where new findings can lead to significant design changes. The development cycle involved repeating phases such as requirements gathering, design, implementation and testing until a satisfactory system was produced.

3.3.1 Requirements Elicitation

Requirements for InSight were gathered through qualitative methods such as stakeholder interviews, surveys and review of related healthcare applications. The primary stakeholders were identified as medical professionals, including ophthalmologists and clinicians, who would use the app to support cataract screening. Preliminary feedback indicated that the app must be simple to use, support image upload (from fundus photographs) and generate results that were accurate, explainable (via Grad-CAM) and exportable for clinical reporting. The elicited requirements were documented to serve as the foundation for subsequent design and development.

3.3.2 Ideation and Planning Phase

The ideation and planning phase involved defining the key features, workflows and technical feasibility of the cataract detection app. This stage focused on outlining core functionalities such as:

- Secure user authentication (Login/Sign Up).
- Role-based access control (Nurse vs. Doctor).

- Fundus image upload and analysis.
- Visual explanation of results (Grad-CAM).
- PDF report export.
- A searchable patient history database.
- A high-level analytics dashboard for doctors.

A roadmap was created to guide development and initial risks (such as data security and model performance) were identified with proposed mitigation strategies (using Supabase for auth and rigorous model validation).

3.3.3 Design Phase

The design phase established the system's structure and clinician-oriented interface as blueprints for implementation. As illustrated in Figure 3.1, development followed an Agile-like process, organized into sprints with iterative reviews to align with clinical workflows.

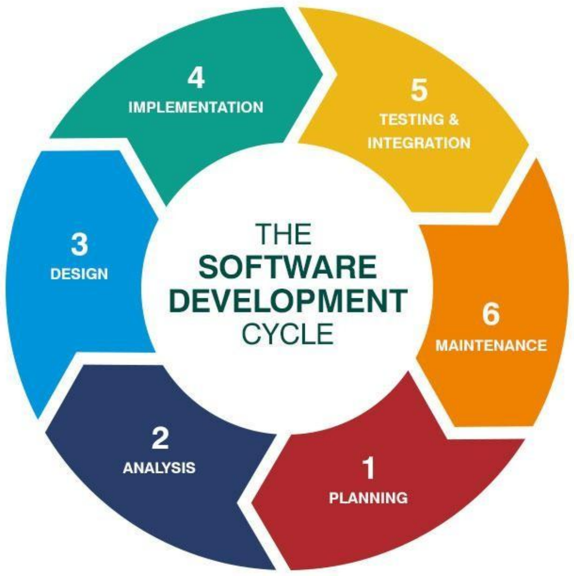


Figure 3.1 Agile Software Development Cycle

Initial wireframes were created in Figma, emphasizing clarity, minimal data entry and intuitive navigation. Architectural diagrams were drafted to define how the Streamlit frontend would communicate securely with the FastAPI backend for AI inference and the Supabase cloud database for user authentication and data storage.

3.3.4 Tool Selection and Setup

The system was built primarily using Python, with a carefully selected tech stack chosen for its modern capabilities and rapid development speed. For the frontend, Streamlit was utilized, as it allowed for the creation of a complex, data-heavy web application purely in Python. The AI backend was powered by FastAPI, which served the ResNet-18 model built with TensorFlow/Keras) through a high-performance API endpoint. For the database and user authentication, Supabase was selected; this backend-as-a-service (BaaS) platform provided a secure PostgreSQL database and managed user authentication, including role-based access.

Key supporting libraries included Plotly for generating analytics charts, FPDF for creating professional PDF reports and Pandas for data manipulation. Throughout the project, Git and GitHub were used for version control. The setup process involved creating virtual environments, managing dependencies and securely configuring API keys using Streamlit's secrets management.

3.3.5 Implementation Planning

The implementation strategy followed a modular approach, which was critical for managing the project's complexity. Each core function, including user authentication, the 'Screening' page, the 'Patient History' page, and the 'Analytics' dashboard, was developed as a distinct, self-contained module within the Streamlit application. This separation of concerns was not only a design choice but a practical necessity. It allowed for focused development on one feature at a time, such as the data-intensive history query or the role-specific logic for the analytics dashboard. This modularity proved invaluable, as it significantly simplified the processes of testing, refinement and debugging, ultimately leading to more robust and maintainable code.

3.3.6 Coding and Development

Coding and development adhered to modern best practices, including consistent naming conventions, constant use of version control and a modular function design as planned. The entire frontend was developed in Python using the Streamlit framework, which enabled the rapid creation of interactive UI components. Concurrently, the AI inference backend was implemented in Python using FastAPI, chosen for its speed and simplicity in wrapping the trained ResNet-18 model. A key component of the architecture was the supabase-py client library. This library was integrated directly into the Streamlit app to handle all database transactions, seamlessly managing user authentication, saving new screening results and fetching patient history records.

3.3.7 Testing and Deployment

The testing strategy for the InSight application was multi-layered. It included unit testing for critical backend components, such as the PDF generation function, to ensure their functional accuracy. More extensive manual integration testing was then performed to verify the entire end-to-end workflow. This testing confirmed that all modules operated seamlessly: from initial user login, through image upload, AI analysis via the FastAPI backend, successful data storage in Supabase and correct retrieval in the patient history page. Deployment was managed by running the Streamlit application and the FastAPI backend service as two distinct, communicating processes. Final validation testing was conducted to assess the system's overall performance, reliability and usability, with brief documentation prepared to guide clinicians in its operation.

3.4 System Analysis and Design

System analysis and design for InSight was focused on ensuring that both the functional flow and technical components were logically and efficiently structured. Since the project utilized distinct services (a web app, an AI microservice and a cloud database), the design prioritized a modular, service-oriented architecture over a single object-oriented (OOP) structure. This approach provided clear separation of concerns, encapsulation and ease of scalability, which were beneficial for handling the complexity of the machine learning-based web application.

3.4.1 Use Case Diagram

The use case diagram outlined how users interact with the system and its main functionalities. The use case for InSight included two primary actors: the Nurse and the Doctor, who both interact with the Streamlit Web Application.

- Nurse (Actor): The Nurse could log in, access the 'Screening' page, upload a patient's fundus image, enter a patient identifier and receive an AI-generated analysis (prediction and Grad-CAM). The Nurse could then save this result to the database and export a PDF report.
- Doctor (Actor): The Doctor had all the permissions of the Nurse. In addition, the Doctor could access the 'Analytics' dashboard to view high-level statistics, charts, and metrics about all screenings performed.

The FastAPI ML model and Supabase Database functioned as key system components that the actors interacted with through the web application, rather than as actors themselves.

3.4.2 Class Diagram

While not a strict object-oriented application, the system's logical design was built around several key entities and components. The User served as a core entity managed by Supabase Auth, which handled all user registration and login, stored credentials, and provided a unique user_id. This was linked to a UserRole, a simple entity in the database that associated a user_id with a text role (Nurse or Doctor) to manage application permissions. The primary data entity was the Screening, which stored all information related to a single scan, including the patient_identifier, predicted_label, prediction_probability, recommendation and the screened_by_user. Interacting with these entities were two key components: the AI_Service, an external FastAPI microservice that encapsulated the ResNet-18 model to provide analysis, and the ReportExporter, a utility module within the Streamlit app that used FPDF to generate PDF reports from Screening data.

3.4.3 Database Schema

The database schema was designed for simplicity, security, and scalability using Supabase (PostgreSQL). It consisted of two custom tables in addition to Supabase's built-in authentication tables. The first, auth.users, was a Supabase-managed table that stored secure user information, including the user_id, email, and encrypted passwords. The second, user_roles, was a custom table that linked a user_id (as a foreign key to auth.users) to a simple role (text), providing an effective way to manage permissions. The third table, screenings, was the main data table. It stored every screening record and contained columns for patient_identifier, predicted_label, prediction_probability, recommendation, screened_by_user, image_filename and a created_at timestamp. Notably, no sensitive patient data (like name or age) was stored, only the non-identifying patient_identifier to maintain privacy.

3.4.4 System Architecture Diagram

The system architecture followed a modern, three-tier web-based design:

1. Presentation Layer (Frontend): This was the Streamlit web application, built entirely in Python. It provided the user interface for all pages (Login, Screening, Patient History, Analytics) and ran in the user's browser.
2. Application Layer (AI Service): This was a separate FastAPI microservice, also built in Python. Its sole responsibility was to host the ResNet-18 model and expose an API endpoint. When the Streamlit app needed an analysis, it sent the image to this service and received a JSON response.

3. Data Layer (Backend/Database): This was provided by Supabase. It handled two distinct jobs: it served as the secure PostgreSQL database for storing user_roles and screenings data, and it managed all user authentication (Login/Sign Up).

This architecture ensured scalability, role-based access, and a clear separation of concerns.

3.4.5 Wireframes

Wireframes served as a visual blueprint of the app's user interface layout and navigation flow. Using Figma, wireframes were created to illustrate the application's visual structure. Key screens included a login/signup page, the 'Screening' page with its upload and results layout and the 'Patient History' page with its search bar and results table. The design emphasized readability, accessibility and ease of use for a clinical environment.

3.5 System Development Tools and Techniques

This section outlines the main tools and techniques that were used in the development of InSight, each chosen based on its suitability for a web-based ML project.

3.5.1 Tool 1 – Python

Python served as the core programming language for the development of InSight. Its simplicity, readability, and extensive support for machine learning and image processing libraries, such as NumPy, OpenCV, and Pandas, made it well-suited for both model development and application logic. Python's versatility was a key factor in its selection, as it was used to build the entire software stack. It powered the machine learning model's training (using TensorFlow/Keras), the high-performance AI backend (using FastAPI) and the interactive frontend (using Streamlit). This eliminated the need for context-switching between different languages and facilitated seamless integration across all components. Its popularity and strong community support also ensured access to a wide range of tutorials and debugging resources, streamlining the development process (Van Rossum & Drake, 2009).

3.5.2 Tool 2 - TensorFlow & TensorFlow Lite

TensorFlow (along with the Keras API) was the core machine learning library used for model development. It was employed to design, train, validate and test the ResNet-18 architecture on the ODIR dataset. The final, trained model was saved and subsequently loaded by the FastAPI backend, which handled the server-side inference. This server-based approach ensured that predictions were fast and consistent, as all computation was performed on the server, removing any reliance on the user's local device (TensorFlow, 2023).

3.5.3 Tool 3 - Figma

Figma was a cloud-based design tool used during the design phase to create high-fidelity wireframes and user interface mockups. It was instrumental in prototyping the clinician workflows for uploading fundus eye images, reviewing AI results with Grad-CAM heatmaps and exporting PDF reports. The collaborative features of Figma supported peer review and iterative improvement of the user interface before development began.

3.5.4 Tool 4 - Git and GitHub

Git was used for local version control and GitHub served as the remote repository. This setup ensured that all development artifacts, from the Streamlit code to the model training scripts, were safely backed up. It also facilitated efficient and transparent development, allowing for the use of feature branches, pull requests and issue tracking, which was crucial for managing the project's milestones.

3.5.5 Technique 1 - Single Model Optimization

Ensemble learning was a technique considered during the initial research phase. While combining outputs from multiple models can sometimes enhance robustness (Dietterich, 2000), the final implementation focused on optimizing a single, efficient ResNet-18 model. This decision was made to prioritize development speed, reduce the complexity of the AI backend, and maintain a clear, direct line of interpretability using Grad-CAM for the single prediction.

3.5.6 Technique 2 - Modular and Functional Programming

Modular programming, rather than strict Object-Oriented Programming (OOP), was applied to structure the InSight application's codebase. The Streamlit application was broken down into reusable and modular components, primarily as distinct Python functions. Each core feature, such as screening_page(), patient_history_page() and create_screening_report(), was encapsulated in its own function. This functional and modular approach promoted a clear separation of concerns, simplified debugging, and ensured the system was maintainable and scalable.

3.5.7 Technique 3 - Iterative Feedback Integration

The development of InSight incorporated iterative feedback integration, where feedback gathered during development and testing was used to refine features and improve usability. This ensured the application remained clinically relevant and addressed real-world workflow needs. By applying these feedback loops throughout the development cycle, the system evolved from its initial wireframes to a functional product that met the core user requirements.

3.6 System Deliverables

3.6.1 System Documentation

System documentation included user guides, setup manuals, model description notes and in-code comments. It supports system understanding, maintenance and reproducibility by providing clear and structured information about the system's architecture, dependencies and operating procedures.

3.6.2 Cataract Prediction Module

This was the core functional component of InSight, implemented as a standalone FastAPI microservice. It took an uploaded fundus photograph as input and ran it through the trained ResNet-18 model to output a JSON response. This response indicated the presence or absence of a cataract, along with a confidence score and the Grad-CAM visualization. This module directly supported the application’s main purpose: assisting clinicians with efficient and reliable cataract detection.

3.6.3 Data Management Module

The data management module was provided by Supabase (PostgreSQL). It handled all secure user authentication (Login/Sign Up) and role-based access control for the Nurse and Doctor. It also provided the structured, cloud-based storage for all application data, primarily the screenings table (storing diagnostic results and metadata) and the user_roles table. This enabled clinicians to review patient cases over time via the 'Patient History' page and supported full traceability.

3.6.4 Visualization and Reporting Module

This module was the Streamlit web application itself, which served as the complete frontend and user interface. It presented analysis results in a clear and interpretable way, including the Grad-CAM heatmaps, textual diagnostic summaries and clinical recommendations. It also provided the critical 'Export PDF Report' functionality (using FPDF) for clinical records. Furthermore, this module included the 'Doctor-only' analytics dashboard, which visualized system-wide metrics. This ensured transparency of AI predictions and supported informed decision-making.

Chapter 4: System Analysis and Design

4.1 Introduction

This chapter details the functional and non-functional requirements that were established during the development of this project, ensuring the final system operates efficiently. The system analysis and design diagrams that were used to structure the application are also included and explained in this chapter.

4.2 System Requirements

Some of the system requirements that were implemented in the project are discussed below:

4.2.1 Functional Requirements

i. Authentication Module: This module was provided by Supabase Auth, which handled secure user registration which is the Sign Up and Login. Supabase automatically managed all password hashing to prevent data breaches. Upon registration, the system sent a verification link to the user's email. The login module compared the user's credentials against the secure auth.users table to ensure only registered users gained access. Upon successful login, the system also queried the custom user_roles table to assign the correct Nurse or Doctor role, which determined their access to features like the Analytics page.

ii. Image Upload Module: This module was the core component of the 'Screening' page. It allowed clinicians to upload patient fundus photographs in formats such as .jpg, .jpeg, or .png format to be interpreted by the CNN model. This functioned as the primary data input for the AI analysis.

iii. Diagnosis Module: Once the uploaded image was processed by the backend AI service, the results were displayed in this module on the 'Screening' page. The output included the predicted result (Cataract Detected or No Cataract Detected), a confidence score, a clinical recommendation and the Grad-CAM heatmap visualization for interpretability.

iv. Records Module: This was implemented as the 'Patient History' page. This module provided a search function where clinicians could enter a patient_identifier. The system then queried the Supabase database and displayed a complete list of all past screening records for that patient, including the screening date, prediction, confidence and the user who performed the scan.

4.2.2 Non-functional Requirements

i. Performance: The system was designed to efficiently support concurrent users. The target average response time was under 7 seconds for model inference and result display, though actual performance depended on factors such as internet connectivity, browser choice and individual device specifications.

ii. Security: Security was ensured through multiple measures. All user passwords were managed by Supabase Auth, which stored them using secure hashing algorithms to protect credentials. Access to the system was controlled via role-based access control (RBAC), ensuring that only authorized medical professionals could log in. This prevented Nurses from accessing the Analytics page. Data transmission was encrypted using HTTPS/TLS protocols by default via the web deployment.

iii. Portability and Compatibility: The system requires a device that can support an internet connection as well as a modern web browser such as Chrome.

iv. Usability: The system was designed to be easy to use for all clinicians, ensuring that it was easy to navigate with minimal training.

4.3 System Analysis Diagrams

4.3.1 Use Case Diagram

A use case diagram is a representation of the relationships between the user of a system and the system itself. It gives information about the different roles that users have in the system. In this particular system, the users were doctors and nurses, whose roles are shown below. Use case diagram for the system is shown in Figure 4.1.

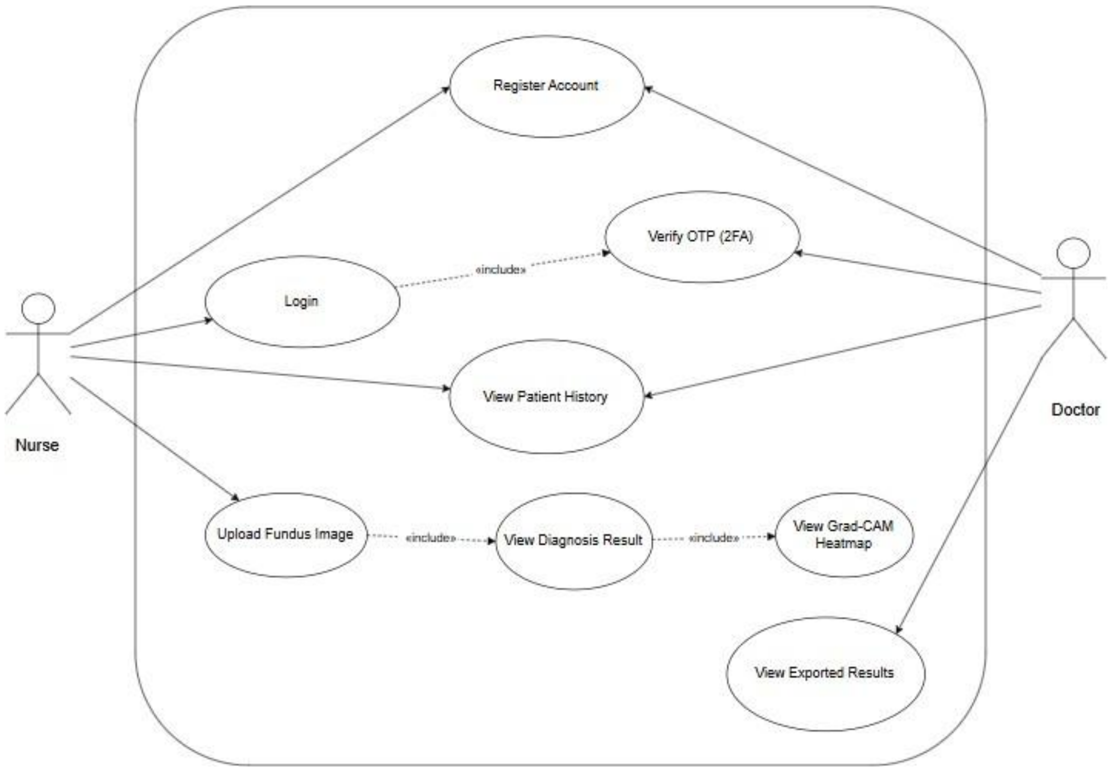


Figure 4.1 Use Case Diagram

4.3.2 Sequence Diagram

A sequence diagram was used to provide a clear representation of the different operations of the system. It shows the order of activities conducted in the system. Sequence diagram for the system is illustrated in Figure 4.2.

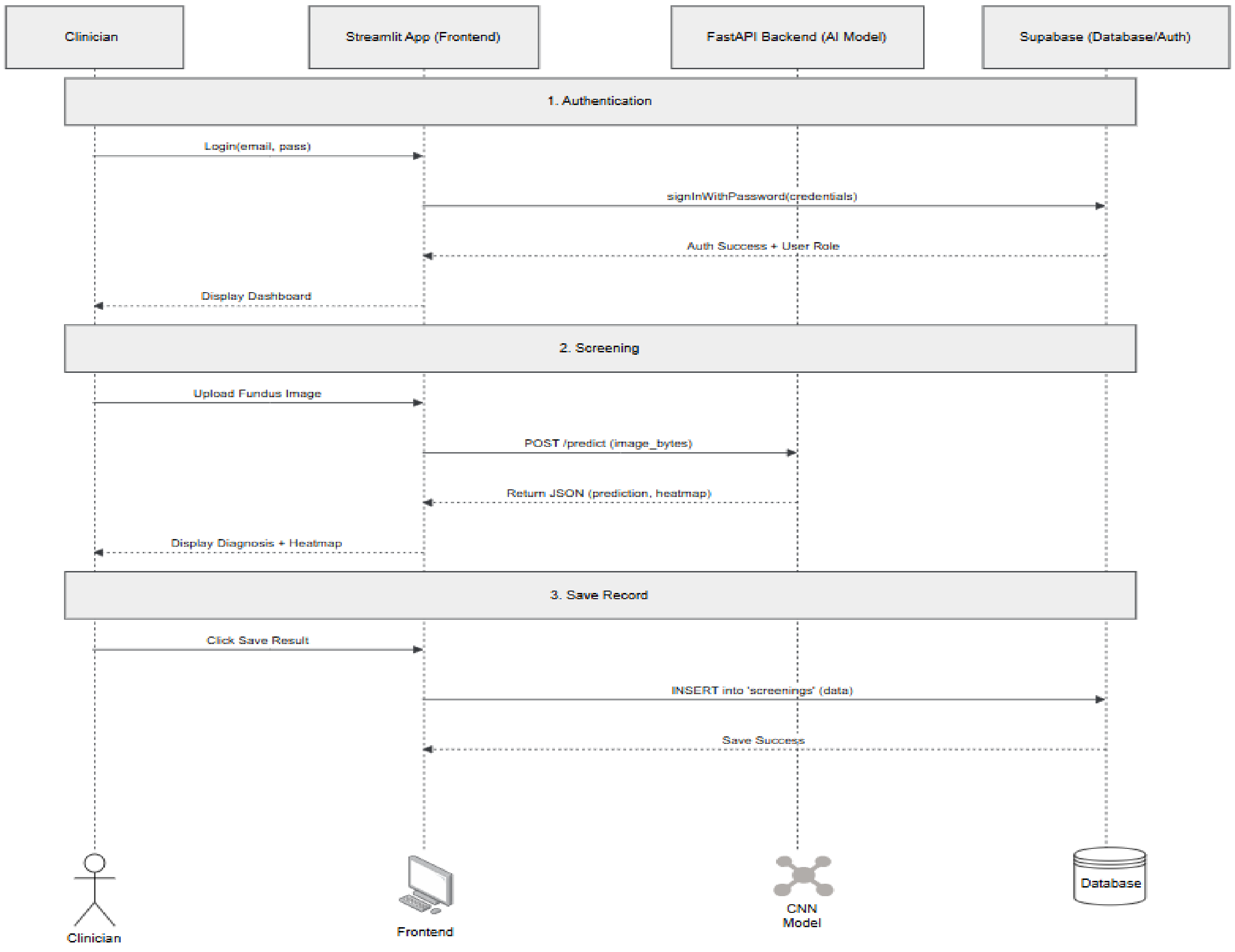


Figure 4.2 Sequence Diagram

4.3.3 Context Diagram

A context diagram was used to show the flow of information between internal and external entities of the system. The context diagram for the system is shown in Figure 4.3.

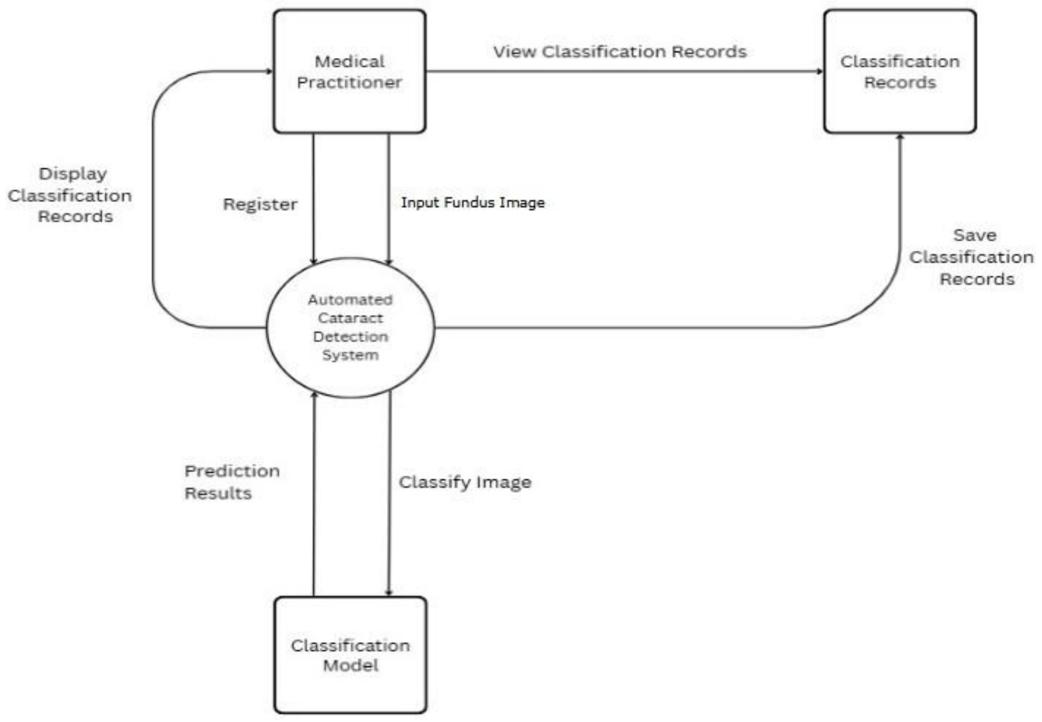


Figure 4.3 Context Diagram

4.3.4 System Class Diagram

Figure 4.4 illustrates the static structure of the system and the relationships between its primary components. The User class manages authentication and maintains a one-to-one relationship with UserRole to enforce role-based permissions (Nurse vs. Doctor). The core Screening entity stores diagnostic data, including the prediction and confidence score, while maintaining a many-to-one relationship with the User. Finally, the AIService and ReportExporter classes encapsulate the functional logic for handling model inference and generating PDF reports, respectively.

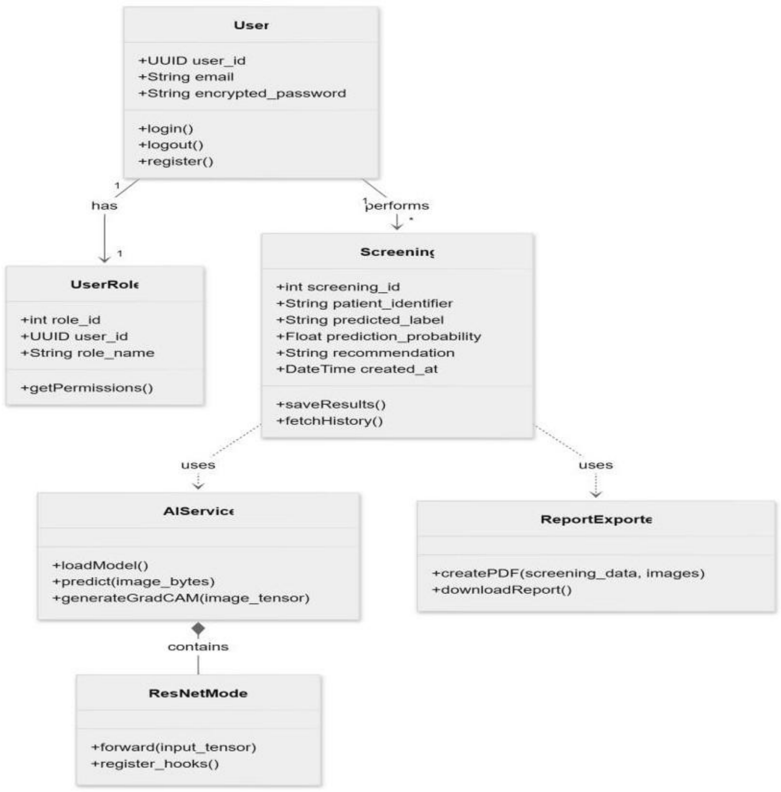


Figure 4.4 System Class Diagram

4.4 System Design Diagrams

4.4.1 Logical Database Schema

A logical database schema was used to design the database before implementing the physical database schema. It is made up of the various tables that make up the system’s database as well as the different keys and attributes in the tables. The database schema for the system is shown in Figure 4.5.

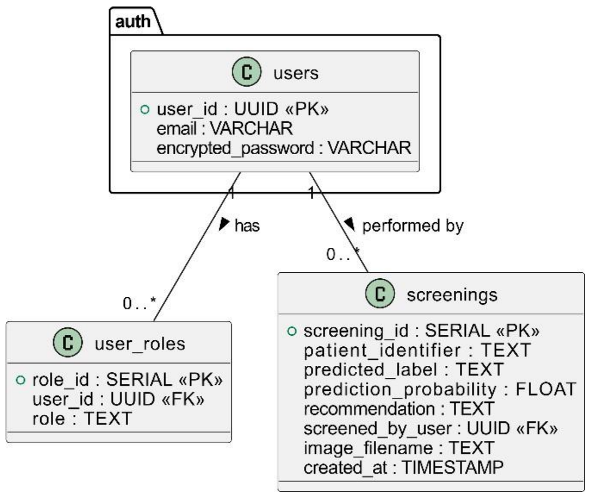


Figure 4.5: Logical Database Schema

4.4.2 System Architecture
A system architecture diagram was used to represent the system's functionality based on the hardware and software components. The system architecture is shown in Figure 4.6.

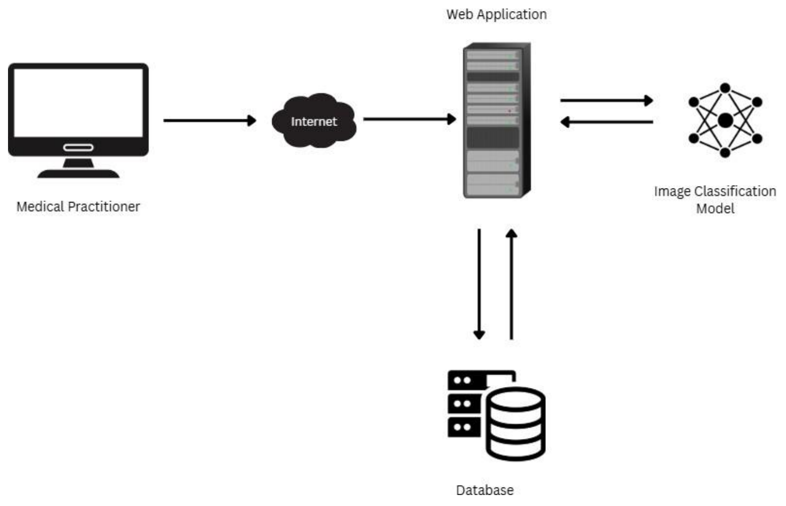


Figure 4.6 System Architecture

4.4.3 System Wireframes
System wireframes were used to visualize the system's user interface before development. The system wireframes are shown in Figures 4.7 and 4.8.

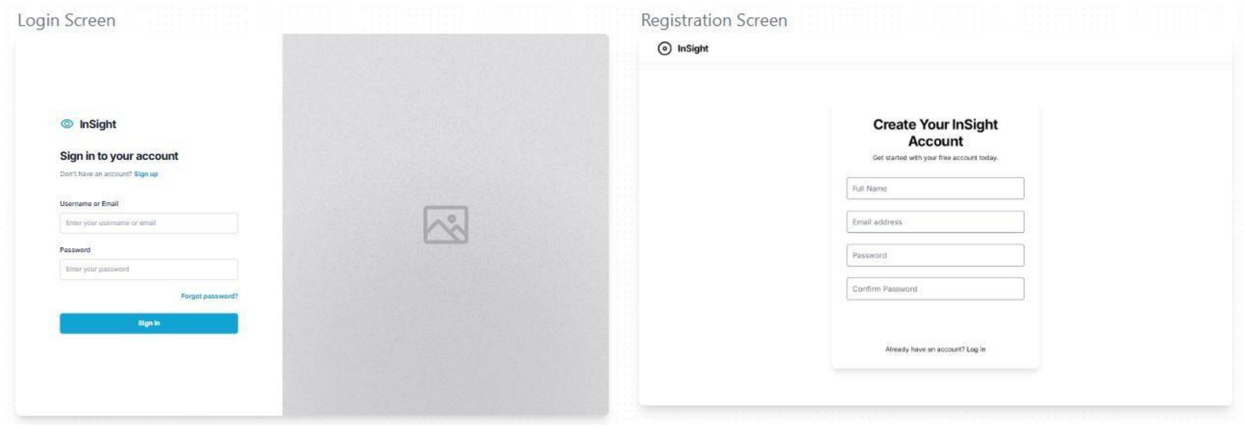
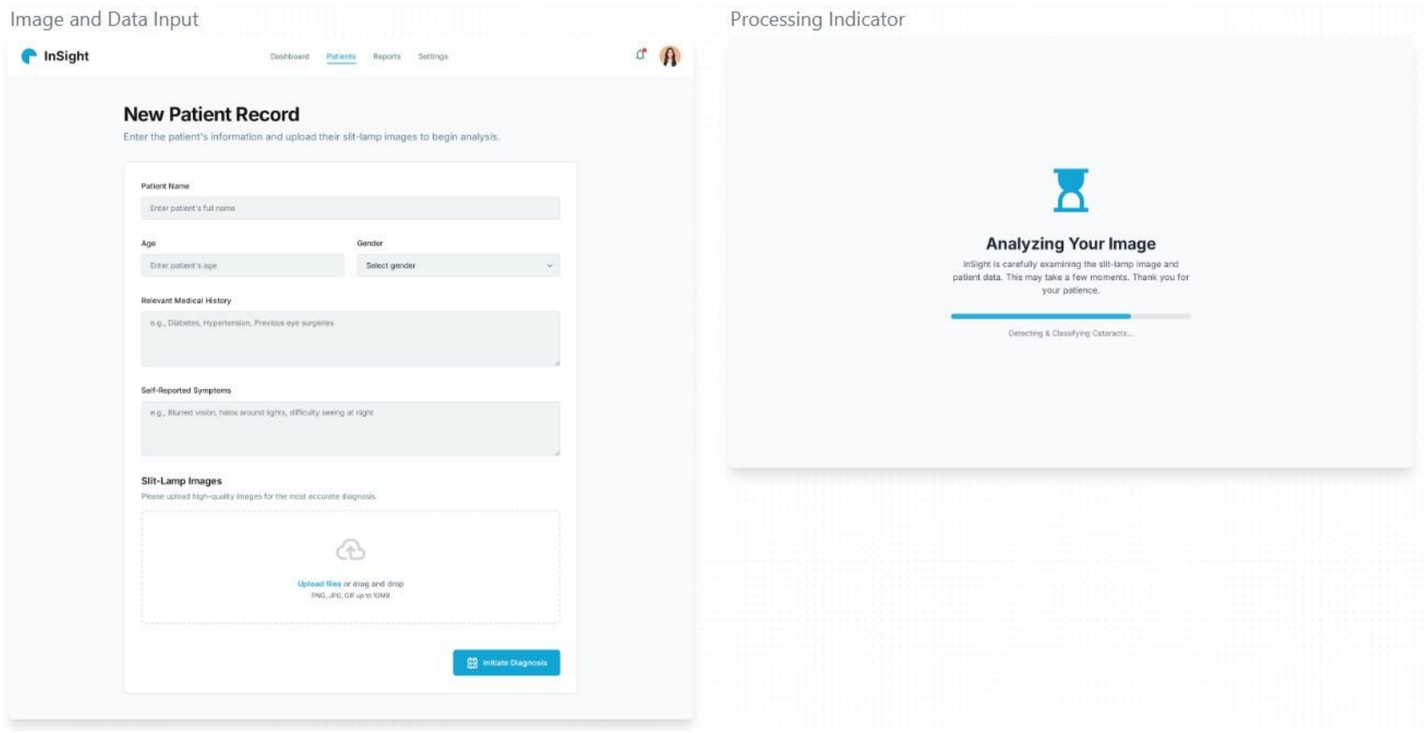


Figure 4.7: Registration and Login Interface Wireframes



Enter the patient's information and upload their slit-lamp images to begin analysis.

Please upload high-quality images for the most accurate diagnosis.

InSight is carefully examining the slit-lamp image and patient data. This may take a few moments. Thank you for your patience.

Figure 4.8: Prediction Interface Wireframe

Chapter 5: System Implementation and Testing

5.1 Introduction

This chapter explains how InSight was implemented and evaluated. It provides a comprehensive overview of the system functionalities, the convolutional neural network classification model and the performance measures used to validate the system. The chapter also details the hardware and software specifications required for both development and production use, along with the implementation environment that ensures optimal system performance.

5.2 Description of the Implementation Environment

The implementation environment consisted of the hardware and software components necessary for training the deep learning model and for developing and running the final application. This section describes the specifications that constituted the system's running environment.

5.2.1 Hardware Specifications

The hardware requirements listed in Table 5.1 below were used for both model training and system development. The training was performed locally using the CPU, which, while slower than a dedicated GPU, was sufficient for achieving a highly accurate model.

Hardware	Justification
16GB RAM	Provided sufficient memory for loading and preprocessing the over 12,000 image dataset in batches and supporting the simultaneous operation of the frontend and backend servers during development.
Intel Core i7 Processor	Handled all model training operations (as a CPU-bound task) and the concurrent processes of the development environment and application servers (FastAPI/Uvicorn, Streamlit).
1 TB SSD	Provided high-speed storage for the large ODIR fundus image dataset, saved model files

	(.pt), the Python virtual environment, and all project code, ensuring fast data access.

Table 5.1: Hardware Specifications

5.2.2 Software Specifications

The software stack, detailed in Table 5.2, was selected to support a modern, full-stack Python application. This architecture allowed for a stable, high-performance backend using FastAPI and an interactive frontend using Streamlit.

Software	Justification
Windows 11	The operating system used for development, providing a stable environment with comprehensive support for Python and all required libraries.
Python 3.11	The primary programming language used for the entire stack, chosen for its extensive machine learning libraries and user-friendly syntax.
PyTorch	The deep learning framework used to train the ResNet18 model. It was chosen for its flexibility and its robust hooks feature, which enabled the successful implementation of Grad-CAM.
FastAPI	A high-performance web framework used to build the backend API. It served the PyTorch model and provided the /predict endpoint that the frontend called.
Streamlit	The web application framework used to build the entire frontend user interface, including the dashboard, login and patient history pages.

Uvicorn	The ASGI server used to run the FastAPI backend, chosen for its high speed and asynchronous capabilities.
Supabase	A cloud-based PostgreSQL service. It was used to manage the user_roles and screenings tables, handle all user authentication and enforce Row Level Security (RLS) policies.

Table 5.2 Software Specifications

5.3 Dataset Description and Preprocessing

The project's success depended on a large, high-quality dataset. The system was trained using the Ocular Disease Intelligent Recognition (ODIR-5K) dataset, a publicly available collection of over 10,000 ocular fundus photographs. This dataset was chosen over smaller slit-lamp datasets to build a more robust and generalizable model.

5.3.1 Ocular Fundus Image Dataset

The ODIR dataset provided a diverse set of real-world clinical images. For this project's binary classification task, the diagnostic keywords for each image were programmatically parsed to create two distinct classes:

- Class 0: No Cataract: Images labeled as 'Normal', 'Diabetes', 'Glaucoma', 'Hypertension', 'Myopia', or 'Other'.
- Class 1: Cataract: Images explicitly containing the 'Cataract' keyword (often alongside other diseases).

After loading the dataset labels and verifying that each image file existed on disk, the final verified dataset used for training consisted of 12,460 fundus images.

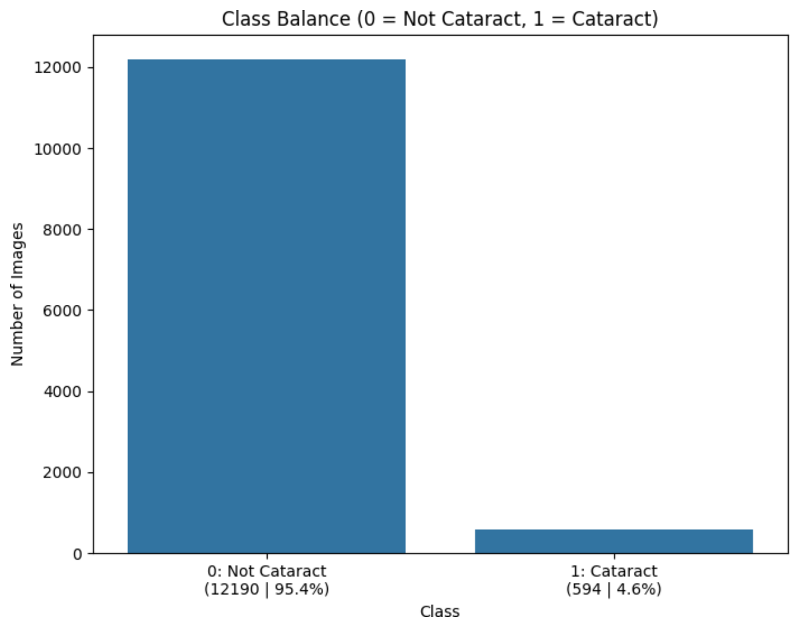


Figure 5.1: Class Distribution of the Verified Fundus Dataset

As shown in Figure 5.1, the dataset was highly imbalanced, with 12,190 No Cataract images and only 594 Cataract images. This 20:1 imbalance presented a significant training challenge, as the model could achieve 95% accuracy by simply guessing No cataract every time. Visual examples of the fundus images included in these categories are presented in Figure 5.2.

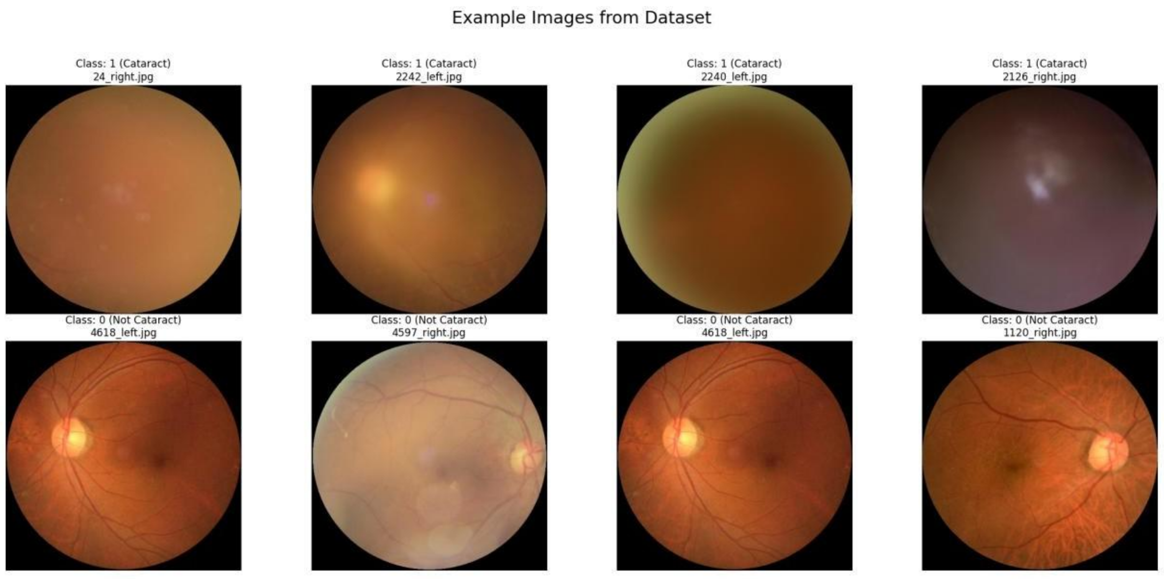

Example Images from Dataset
Class: 1 (Cataract) 24_right.jpg
Class: 1 (Cataract) 2242_left.jpg
Class: 1 (Cataract) 2240_left.jpg
Class: 1 (Cataract) 2126_right.jpg
Class: 0 (Not Cataract) 4618_left.jpg
Class: 0 (Not Cataract) 4597_right.jpg
Class: 0 (Not Cataract) 4618_left.jpg
Class: 0 (Not Cataract) 1120_right.jpg

Figure 5.2: Example of Fundus Images from the Dataset

5.3.2 Image Preprocessing and Augmentation Strategy

All images were processed using a `torchvision.transforms` pipeline. For the validation set, images were simply resized to 224x224 pixels, converted to a tensor and normalized using standard ImageNet statistics, as required by the pre-trained ResNet18 model.

To address the severe class imbalance and prevent overfitting, the training set used a more aggressive data augmentation pipeline. This pipeline created new, varied examples in real-time during training, teaching the model what a cataract looks like from different angles and in different lighting conditions.

The `train_transform` pipeline included:

- Resize: All images resized to 224x224 pixels.
- RandomHorizontalFlip: 50% chance of flipping the image horizontally.
- RandomRotation: Randomly rotated the image by up to 15 degrees.
- ColorJitter: Randomly altered the brightness and contrast of the image.
- ToTensor & Normalize: Converted the image to a tensor and normalized its pixel values.

5.3.3 Dataset Splitting

The verified dataset of 12,460 images was partitioned using an 80% train / 20% validation split. This was done stratified, meaning the 20:1 ratio of normal-to-cataract images was preserved in both the training and validation sets and this ensured that the model was tested on a realistic data distribution.

- Training Set: 9,968 images (used for training the model)
- Validation Set: 2,492 images (used to evaluate the model's performance after each epoch and save the best version)

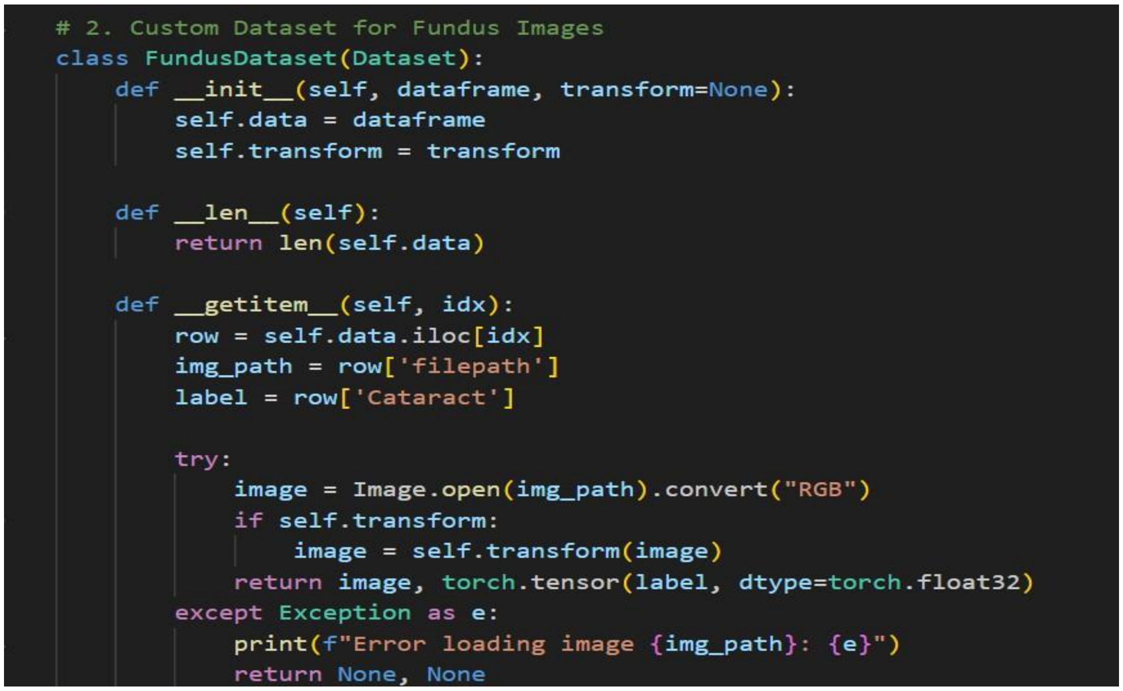
5.4 Model Architecture and Training

The implementation of the machine learning pipeline was a multi-stage process involving data preparation, model selection and training.

5.4.1 Data Loading and Preprocessing

Unlike a simple directory structure, the ODIR dataset required custom processing. A Python script was developed to read the Comma Separated Values(csv) file, which contained the filenames and diagnostic keywords. This script parsed the keywords to create the binary label (Cataract=1, No Cataract=0) and verified that each image file existed on disk.

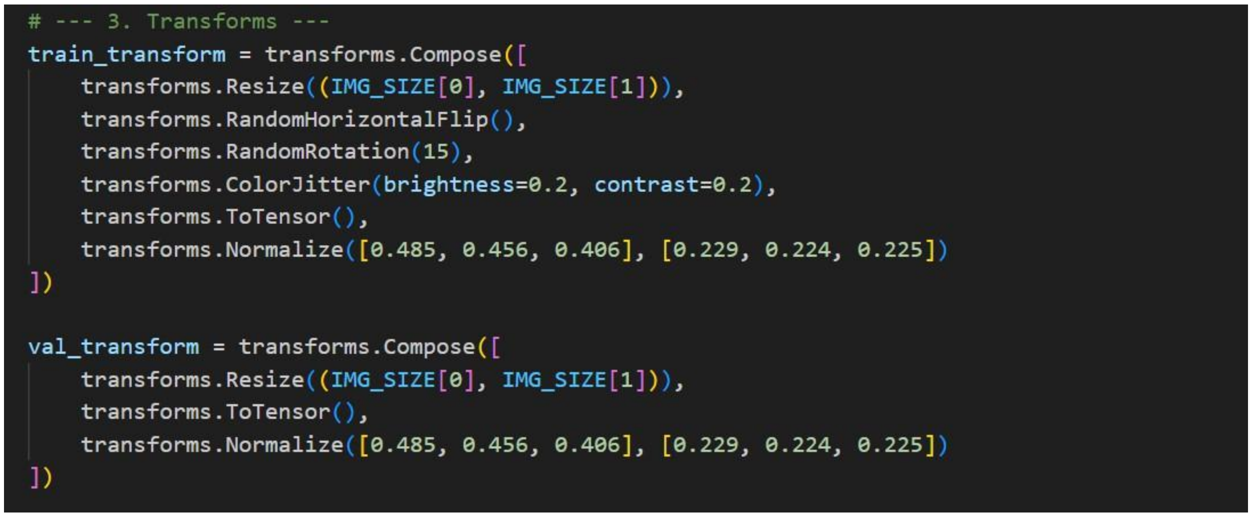
A custom PyTorch Dataset class, named FundusDataset, was implemented as shown in Figure 5.3. This class was responsible for loading an image from its file path, converting it to RGB as well as applying a set of transformations.



```python
# 2. Custom Dataset for Fundus Images
class FundusDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.data = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        row = self.data.iloc[idx]
        img_path = row['filepath']
        label = row['Cataract']

        try:
            image = Image.open(img_path).convert("RGB")
            if self.transform:
                image = self.transform(image)
            return image, torch.tensor(label, dtype=torch.float32)
        except Exception as e:
            print(f"Error loading image {img_path}: {e}")
            return None, None
```

Figure 5.3: Custom PyTorch FundusDataset Class

To prepare the data for the pre-trained ResNet18 model, two distinct transformation pipelines were created using torchvision.transforms, as illustrated in Figure 5.4. The val_transform (for validation data) only resized and normalized the images. The train_transform (for training data) included significant data augmentation to prevent overfitting and teach the model to recognize cataracts under various conditions. This augmentation created thousands of unique image variations during training.



```
# --- 3. Transforms ---
train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE[0], IMG_SIZE[1])),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE[0], IMG_SIZE[1])),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

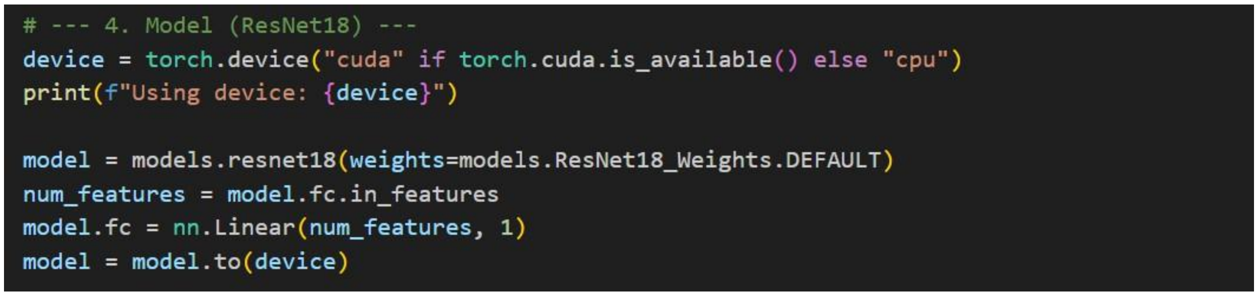
Figure 5.4: Training and Validation Image Transforms

Finally, PyTorch DataLoaders were used to feed this data to the model in batches of 32. A custom collate_fn was implemented to safely skip any corrupted image files that failed to load, preventing training from crashing.

5.4.2 Model Architecture and Training Setup

A Transfer Learning approach was selected for this project. Instead of building a CNN from scratch, the ResNet18 model, pre-trained on the ImageNet dataset, was used. This approach leverages a powerful, proven architecture, resulting in faster convergence and higher accuracy.

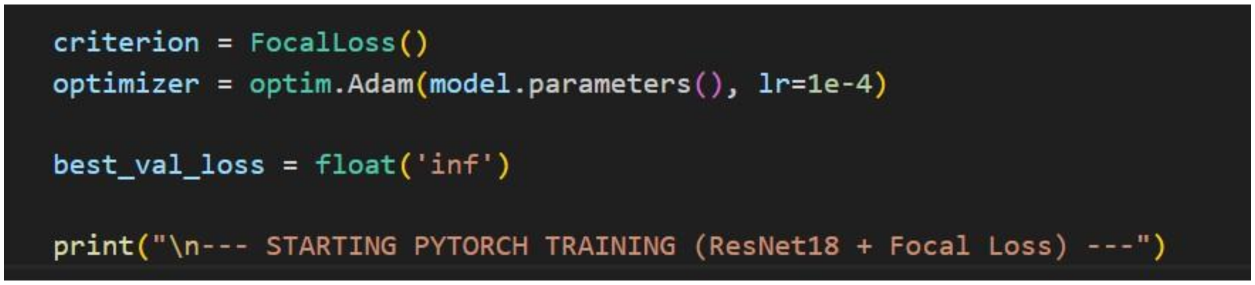
The model was loaded using models.resnet18(weights=models.ResNet18_Weights.DEFAULT). The final fully connected (fc) layer, which was originally designed to classify 1,000 ImageNet classes, was removed and replaced with a new torch.nn.Linear layer with a single output (1 logit). This adapted the model for our binary (Cataract vs. No Cataract) classification task as shown in Figure 5.5.



```
# --- 4. Model (ResNet18) ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 1)
model = model.to(device)
```

Figure 5.5: ResNet18 Model Definition

To combat the severe 20:1 class imbalance, a simple class_weight was insufficient. Instead, a Focal Loss function was implemented. This specialized loss function automatically focused the model's attention on hard-to-classify examples (the rare cataract cases) and down-weighted the loss from the thousands of easy, common No Cataract images. This prevented the model from becoming biased. The model was compiled using the Adam optimizer as illustrated in Figure 5.6.



```
criterion = FocalLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)

best_val_loss = float('inf')

print("\n--- STARTING PYTORCH TRAINING (ResNet18 + Focal Loss) ---")
```

Figure 5.6: Focal Loss and Optimizer Setup

5.4.3 Model Training and Results

The ResNet18 model was trained on the 9,968 training images for up to 30 epochs using the CPU. The training script was configured to monitor the validation loss after every epoch. The model's state was saved to the file fundus_pytorch_model.pt only when the validation loss on the 2,492 validation images improved.

The training ran for 30 epochs, with the best model being saved at Epoch 27, which achieved a validation accuracy of 99.52% and a best validation loss of 0.0006. This extremely high accuracy and low loss demonstrated that the combination of ResNet18, a large dataset and Focal Loss was highly effective.

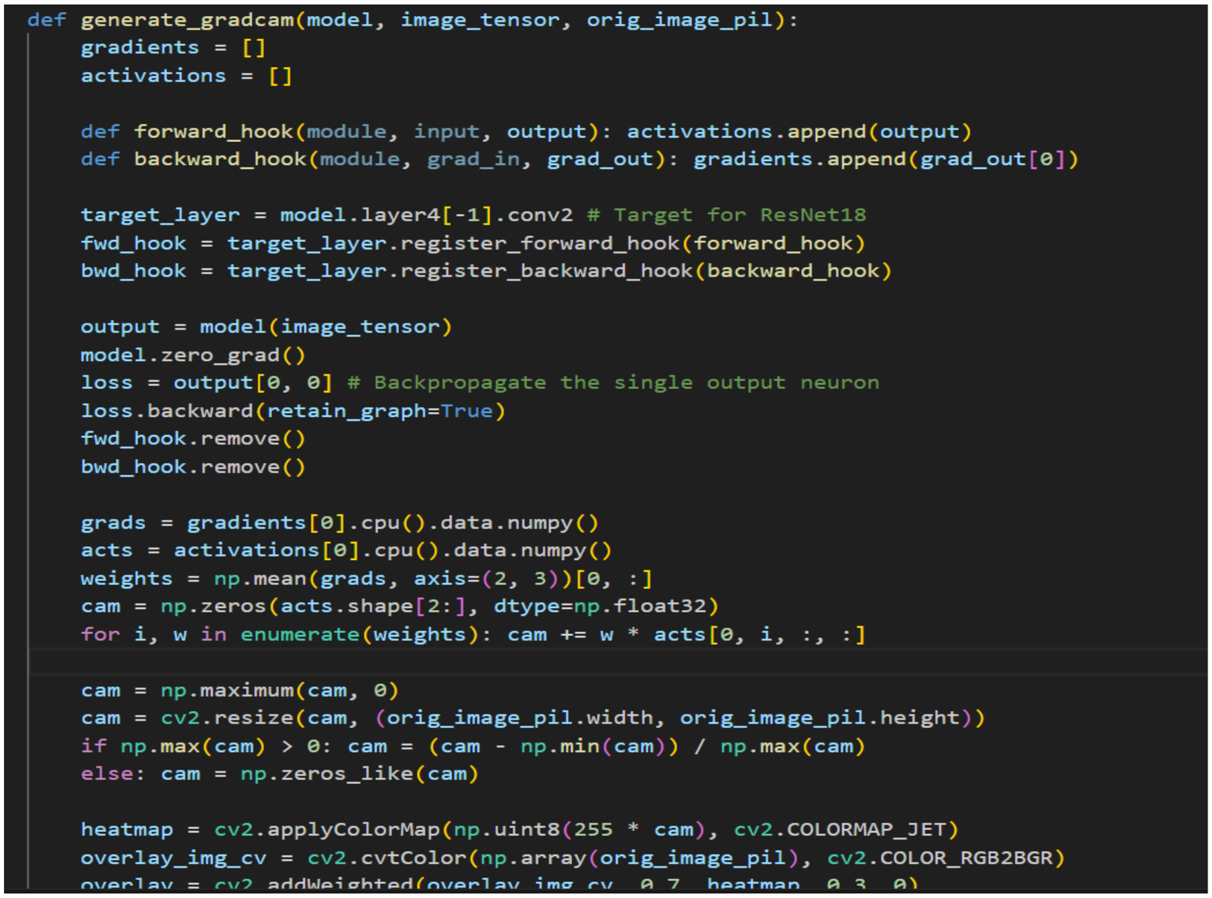
5.4.4 Grad-CAM Explainability Implementation

A critical feature of InSight was AI explainability. To avoid the graph-related errors encountered with TensorFlow, this project used the PyTorch hooks method within the FastAPI backend. A function, generate_gradcam, was created in backend_app.py. When a user requested a prediction, this function:

1. Attached a register_forward_hook to the final convolutional layer of the ResNet18 model (model.layer4[-1].conv2) to capture its activation maps.
2. Attached a register_backward_hook to the same layer to capture the gradients.
3. Performed a backward pass from the model's final output to calculate the gradients.

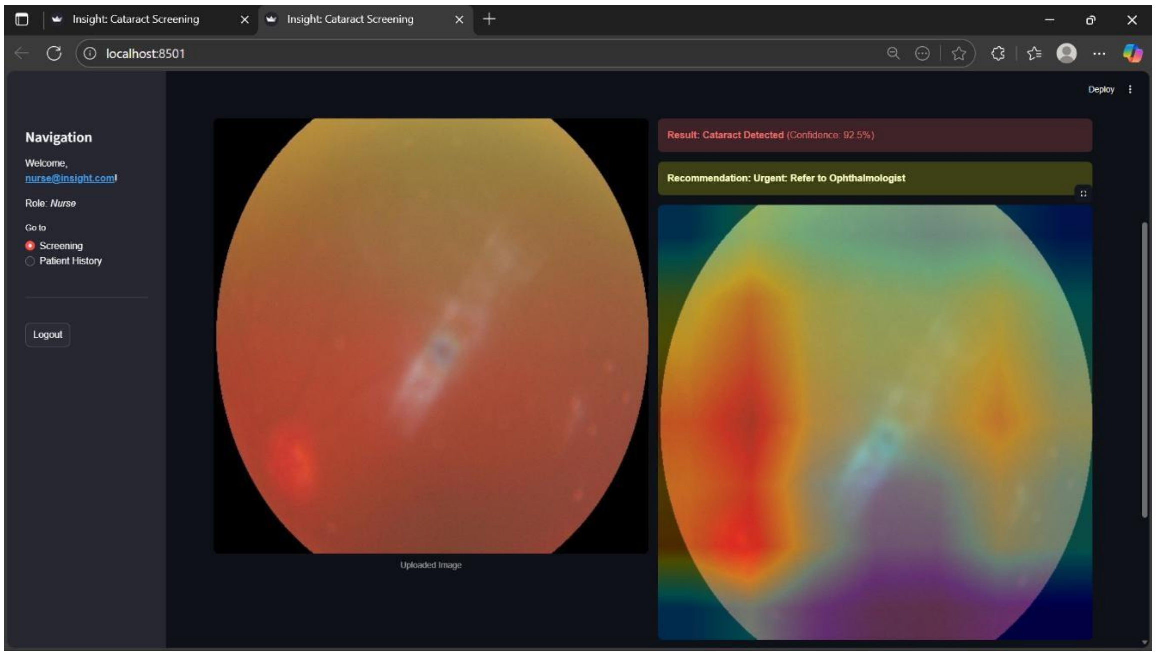
4. Used the gradients to calculate the importance of each activation map.
5. Combined these weighted maps into a single heatmap, which was then resized and overlaid onto the original image using OpenCV (cv2).

This entire process ran on the backend server as illustrated in Figure 5.7, which then sent the final Grad-CAM image back to the Streamlit frontend for display resulting in the interface shown in Figure 5.8.



```python
def generate_gradcam(model, image_tensor, orig_image_pil):
    gradients = []
    activations = []

    def forward_hook(module, input, output): activations.append(output)
    def backward_hook(module, grad_in, grad_out): gradients.append(grad_out[0])

    target_layer = model.layer4[-1].conv2 # Target for ResNet18
    fwd_hook = target_layer.register_forward_hook(forward_hook)
    bwd_hook = target_layer.register_backward_hook(backward_hook)

    output = model(image_tensor)
    model.zero_grad()
    loss = output[0, 0] # Backpropagate the single output neuron
    loss.backward(retain_graph=True)
    fwd_hook.remove()
    bwd_hook.remove()

    grads = gradients[0].cpu().data.numpy()
    acts = activations[0].cpu().data.numpy()
    weights = np.mean(grads, axis=(2, 3))[0, :]
    cam = np.zeros(acts.shape[2:], dtype=np.float32)
    for i, w in enumerate(weights): cam += w * acts[0, i, :, :]

    cam = np.maximum(cam, 0)
    cam = cv2.resize(cam, (orig_image_pil.width, orig_image_pil.height))
    if np.max(cam) > 0: cam = (cam - np.min(cam)) / np.max(cam)
    else: cam = np.zeros_like(cam)

    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)
    overlay_img_cv = cv2.cvtColor(np.array(orig_image_pil), cv2.COLOR_RGB2BGR)
    overlay = cv2.addWeighted(overlay_img_cv, 0.7, heatmap, 0.3, 0)
```

Figure 5.7: Grad-CAM Implementation using PyTorch Hooks



Result: Cataract Detected (Confidence: 92.5%)

Recommendation: Urgent: Refer to Ophthalmologist

Figure 5.8: Final System Output (Prediction with Grad-CAM)

5.5 System Implementation

The InSight system was implemented as a full-stack, client-server application to ensure stability, scalability and a clear separation of concerns. This architecture was comprised of two primary components: a Python FastAPI backend and a Python Streamlit frontend.

5.5.1 Web Application Architecture

The FastAPI backend was developed to serve as the project's computational engine. Its sole responsibilities were to load the trained PyTorch ResNet18 model (fundus_pytorch_model.pt) into memory and expose a secure /predict endpoint. This backend handled all computationally intensive tasks, including image preprocessing, model inference and the generation of Grad-CAM visualizations using PyTorch's hook-based method. This design ensured that the user interface remained fast and responsive, as it offloaded all heavy processing to a separate, dedicated server.

The Streamlit frontend was built to function as the complete, interactive user interface. It managed all user-facing logic, including secure login and sign-up pages that communicated directly with the Supabase authentication service. It coordinated the clinical workflow by sending the uploaded image data to the FastAPI backend for analysis. The frontend was also responsible for receiving the backend's JSON response (containing the prediction and Grad-CAM image) and securely communicating with the Supabase database. This allowed it to save screening results to the screenings table and query data for the Patient History and Analytics pages, enforcing all security rules based on the user's role.

This architecture supported distinct clinical workflows based on role-based access, managed in the user_roles Supabase table. The Nurse interface focused on the primary screening loop (upload, analyze, save and export), while the Doctor interface provided additional, privileged access to the Analytics dashboard for comprehensive oversight.

5.5.2 Workflow Integration

The InSight system implemented an end-to-end clinical workflow that began with secure role-based authentication. A healthcare provider such as a nurse logged in via the Streamlit interface, which verified their credentials and role with Supabase.

On the New Patient Screening page, the nurse uploaded an ocular fundus image and entered a patient identifier. Upon clicking Analyze Image, the Streamlit frontend sent the image data via an API call to the FastAPI backend's /predict endpoint. The backend then executed the core diagnostic phase: it preprocessed the image, ran the PyTorch model to generate a prediction probability and concurrently produced the Grad-CAM visualization. This entire package comprising of the probability and the Grad-CAM image was then sent back to the frontend in a single JSON response.

The Streamlit interface immediately displayed the results to the user. This included a clear diagnostic outcome (Cataract Detected or No Cataract Detected) based on the model's confidence, a clinical recommendation (Urgent or Routine), and the Grad-CAM heatmap for visual interpretation. From there, the clinician had three options: Save Results, which logged the screening details to the Supabase screenings table; Export Grad-CAM, to save the heatmap image; or Export PDF Report. This final option generated a professional, shareable report containing all patient and screening details, as well as side-by-side images of the original fundus photo and its corresponding Grad-CAM overlay for clear clinical reference. This workflow successfully concluded by creating a secure, persistent record that could then be accessed via the Patient History or Analytics pages, completing the data's lifecycle.

5.6 Testing and Evaluation

The system underwent a rigorous, two-part evaluation. First, the PyTorch model's predictive performance was quantitatively assessed during and after training. Second, the full-stack web application was subjected to a series of functional tests to validate the complete, end-to-end clinical workflow, from user login to report generation.

5.6.1 Performance Metrics

The model's performance was evaluated using a comprehensive suite of metrics specifically selected for clinical relevance in medical diagnostic applications. Each metric was chosen to address different aspects of diagnostic performance:

- Accuracy served as the fundamental measure of overall prediction correctness, representing the proportion of true results (both true positives and true negatives) among the total number of cases examined.
- Loss (Focal Loss) was the primary metric optimized during training. A decreasing validation loss indicated that the model was becoming more confident and accurate in its predictions on unseen data.
- Precision (Positive Predictive Value) was monitored to measure the model's reliability in cataract identification, quantifying the proportion of true cataract cases among all cases predicted as Cataract. This was critical for minimizing false positives.
- Recall (Sensitivity) was prioritized to ensure the system effectively identified genuine cataract cases. This metric measured the model's ability to detect actual cataract conditions, with high recall being clinically crucial to prevent false negatives.
- The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) was used as a comprehensive assessment of the model's discrimination capability across all classification thresholds, with values approaching 1.0 indicating excellent diagnostic performance.

5.6.2 Model Performance Evaluation

The PyTorch ResNet18 model was trained on the 9,968 images in the training set and evaluated against the 2,492 images in the validation set. Training was conducted for 30 epochs using the Focal Loss function, which proved highly effective in managing the dataset's 20:1 class imbalance.

A ModelCheckpoint callback was implemented to monitor the val_loss (validation loss) at the end of each epoch. The model's state dictionary (fundus_pytorch_model.pt) was saved to disk only when the model achieved a new best (lowest) validation loss.

The training was extremely successful. The model converged quickly, and the val_loss steadily decreased. The best model was saved from Epoch 27, which achieved an exceptionally low validation loss of 0.0006 and a validation accuracy of 99.52%. This demonstrated that the model had successfully learned to distinguish cataracts from healthy fundus images with a very high degree of reliability.

5.6.3 Functional System Testing

After the backend and frontend servers were deployed locally, a series of functional tests were performed to validate the entire application workflow from end to end. The specific test cases and their results are detailed in Table 5.3 below.

Test Case ID	Test Description	Expected Result	Actual Result	Status
TC-001	User Authentication: Attempt to log in with valid credentials via the Streamlit interface.	System verifies credentials with Supabase Auth and grants access to the dashboard.	Login successful; user redirected to the main dashboard.	Pass
TC-002	Role-Based Access (Nurse): Log in as a Nurse and attempt to access the Analytics page.	Access to Analytics is restricted/hidden. User only sees Screening and Patient History.	Analytics page was hidden from the navigation menu.	Pass
TC-003	Role-Based Access (Doctor): Log in as a Doctor and attempt to	Analytics dashboard is visible and accessible.	Doctor successfully accessed the Analytics dashboard	Pass

	access the Analytics page.		with charts displayed.	
TC-004	Cataract Prediction Pipeline: Upload a known Cataract fundus image and click Analyze.	System returns Cataract Detected with high confidence and displays a Grad-CAM heatmap.	Prediction: Cataract Detected(Confidence > 90%). Heatmap correctly highlighted the lens opacity.	Pass
TC-005	Normal Eye Prediction Pipeline: Upload a known Normal fundus image and click Analyze.	System returns No Cataract Detected and displays a Grad-CAM heatmap.	Prediction: No Cataract Detected. Heatmap generated successfully.	Pass
TC-006	Data Persistence: Click Save Results after a screening.	Screening data (Patient ID, Prediction, Date) is inserted into the Supabase screenings table.	New record immediately appeared in the Supabase database table.	Pass
TC-007	Patient History Retrieval: Search for the Patient ID from TC-006 in the Patient History page.	The saved record should appear in the results table.	Record was successfully retrieved and displayed in the history table.	Pass
TC-008	Report Generation: Click Export PDF Report for a completed screening.	A PDF file containing patient details, diagnosis and side-by-side images is	PDF was generated correctly with all details and images included.	Pass

		generated and downloaded.		

Table 5.3: Functional System Test Cases

Chapter 6: Conclusions, Recommendations and Future Works

6.1 Conclusion

It can be concluded that a Convolutional Neural Network (CNN), specifically a ResNet18 architecture, can be used to highly automate and accurately perform the process of detecting cataracts from ocular fundus images. The developed solution, InSight, successfully achieved its primary objectives. The trained PyTorch model reached a validation accuracy of 99.52% with a best validation loss of 0.0006, demonstrating that the use of a Focal Loss function was an effective strategy to counteract the severe 20:1 class imbalance in the training data.

The project also concluded that a two-part, client-server architecture (Streamlit frontend and FastAPI backend) was a superior solution to a monolithic design. This separation of concerns was the key to successfully implementing advanced features; it stabilized the PyTorch model's execution and enabled the PyTorch hooks method for Grad-CAM to run reliably, a feature that had previously failed in an integrated TensorFlow/Streamlit environment.

The final system delivered a complete clinical workflow, integrating secure role-based authentication, a patient history lookup feature, and an analytics dashboard, all connected to a persistent Supabase database. This confirmed that it was feasible to build a tool that not only provided highly accurate predictions but also met the practical needs of a clinical setting, streamlining triage and providing a foundation for reducing diagnostic delays.

6.2 Recommendations

To ensure the continued improvement and real-world deployment of the developed solution, several recommendations are proposed. First, expanding the minority class data is essential. While the model has demonstrated high performance, its robustness could be further enhanced. Beyond relying solely on loss-function techniques such as Focal Loss, acquiring additional verified positive Cataract fundus images is recommended. A more balanced dataset would improve the model’s ability to identify borderline cases and increase its generalizability to diverse patient populations. In addition, support for the DICOM file format should be added. Currently, the application accepts standard web image formats like PNG and JPG. To enable full integration with hospital PACS systems, the FastAPI backend should be extended to read and process medical-standard DICOM (.dcm) files, which often contain richer patient metadata.

Finally, in-app role management should be implemented. At present, assigning the “Doctor” role requires a developer to manually update the Supabase database. It is recommended to develop a secure Admin interface within the Streamlit application, allowing lead clinicians to manage user roles directly without requiring database access.

6.3 Future Works

In the future, this project lays a strong foundation for several important enhancements. One potential direction is the development of a multi-class severity grading system. Rather than simply detecting the presence of a cataract, the model could classify its severity, such as mild, moderate, or severe, offering greater clinical value for triage and surgical planning. Another possibility is expanding the model to perform multi-label disease detection. Leveraging the labels in the ODIR dataset, the system could simultaneously screen for cataracts, glaucoma, and diabetic retinopathy from a single fundus image, thereby maximizing the utility of each screening. Additionally, alternative model architectures could be explored. Implementing advanced networks like EfficientNetV2 or Vision Transformers (ViT) would allow comparisons in performance, inference speed, and the quality of Grad-CAM visualizations. Finally, for real-world clinical adoption, the FastAPI backend could be enhanced to integrate with hospital Electronic Health Record (EHR) systems using standards such as HL7 or FHIR, enabling screening results to be automatically and securely recorded in a patient’s official file.

References

Allen, L. N., Karanja, S., Gichangi, M., Bunywera, C., Rono, H., Macleod, D., Kim, M. J., Tlhajoane, M., Burton, M. J., Ramke, J., & Bastawrous, A. (2024). Access to community-based eye services in Meru, Kenya: A cross-sectional equity analysis. *International Journal for Equity in Health, 23*, 170.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *International Workshop on Multiple Classifier Systems*, 1–15. Springer, Berlin, Heidelberg

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Gutierrez, L. J., Lim, J. S., Foo, L. L., Ng, W. Y., Yip, M., Lim, G. Y. S., Wong, M. H. Y., Fong, A., Rosman, M., Mehta, J. S., Lin, H., Ting, D. S. J., & Ting, D. S. W. (2022). Application of artificial intelligence in cataract management: current and future directions. *Eye and Vision (London), 9*, 11.

Gadde, S. G., et al. (2024). "Barriers to follow-up care in mobile eye health initiatives: A cross-sectional study in East Africa." Journal of Global Health, 14(2), 45-51.

Mayo Clinic. (2014). *Cataract*. [Illustration]. Mayo Foundation for Medical Education and Research.

Mwangi, N., et al. (2024). "Slit-lamp accessibility in Sub-Saharan Africa: A supply-chain analysis." Ophthalmic Epidemiology, 31(1), 12-19.

Okeke, S., et al. (2023). "Ophthalmologist workforce gaps in Kenya: A geospatial mapping study." Lancet Global Health, 11(6), e892-e900.

Patel, R., et al. (2023). "Delays in cataract presentation: Qualitative insights from Nakuru County, Kenya." BMJ Open Ophthalmology, 8(1), e001233.

Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach* (7th ed.). McGraw-Hill Higher Education.

Rono, H. K., et al. (2023). "Smartphone-based cataract screening by non-specialists: Image quality challenges." Digital Health, 9, 1-9.

Peking University. (2019). *ODIR-2019: Grand Challenge on Ocular Disease Intelligent Recognition*.

Bastawrous, A., et al. (2023). "Mobile AI for community cataract screening." Digital Medicine, 6(1), 45.

Li, X., Hu, Y., Chen, H., Wang, J., & Liu, Y. (2020). Ensemble deep learning for slit-lamp image–based cataract detection and grading using Lens Opacities Classification System III. *Ophthalmology and Therapy, 9*(3), 1–13.

Raghavendra, U., Rao, A., Swarnalatha, R., & Gudlavalleti, V. S. (2018). Automated cataract detection using retinal fundus images for population screening. *Eye, 32*(6), 1134–1140.

Shimizu, E., Tanji, M., Nakayama, S., Ishikawa, T., Agata, N., Yokoiwa, R., Nishimura, H., Khemlani, R. J., Sato, S., Hanyuda, A., & Sato, Y. (2023). AI-based diagnosis of nuclear cataract from slit-lamp videos. *Scientific Reports, 13*, 22046.

Tham, Y.-C., Goh, J. H. L., Anees, A., Lei, X., Rim, T. H., Chee, M.-L., Cheung, N., Hamzah, H., Tan, G. S. W., Husain, R., Sabanayagam, C., Liu, Y., Wong, T. Y., & Cheng, C.-Y. (2022). Detecting visually significant cataract using retinal photograph-based deep learning. *Nature Aging, 2*, 242–251.

Triningrat, A. A. M. P., Doniho, A., Jayanegara, W. G., Widiana, I. G. R., Wigono, S., Utari, N. M. L., Shimizu, E., Nakayama, S., & Foraldy, T. (2025). Reliability and accuracy of a smart eye camera in determining grading of nuclear cataract. *Korean Journal of Ophthalmology, 39*(2), 114–124.

Vision Loss Expert Group of the Global Burden of Disease Study. (2023). Global estimates on the number of people blind or visually impaired by cataract: A meta-analysis from 2000 to 2020. *Eye, 38*, 2156–2172.

Tan, N., et al. (2024). "Ensemble models for comprehensive cataract management." Journal of AI in Medicine, 5(3), 205-215.

TensorFlow. (2023). *TensorFlow: Large-scale machine learning on heterogeneous systems* [Software].

Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Berkeley, CA: Apress.

Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Python Software Foundation.

Liskov, B. (1987). Data abstraction and hierarchy. ACM SIGPLAN Notices, 23(5), 17–34.

Raghavendra, U., Fujita, H., Bhandary, S. V., Gudigar, A., Tan, J. H., & Acharya, U. R. (2018). Deep convolution neural network for accurate diagnosis of glaucoma using digital fundus images. *Information Sciences*, *441*, 41-49.

Padalia, D., Mazumdar, A., & Singh, B. (2022). A CNN-LSTM combination network for cataract detection using eye fundus images. *arXiv preprint arXiv:2210.16093*.

Appendix

Appendix 1: Gantt Chart

The Gantt chart below is used to visually track the progress of the InSight project across its various phases. It highlights key milestones, timelines and deadlines, helping to manage tasks efficiently and ensure timely completion. This chart also served as a planning tool to monitor dependencies and progress throughout the development process.

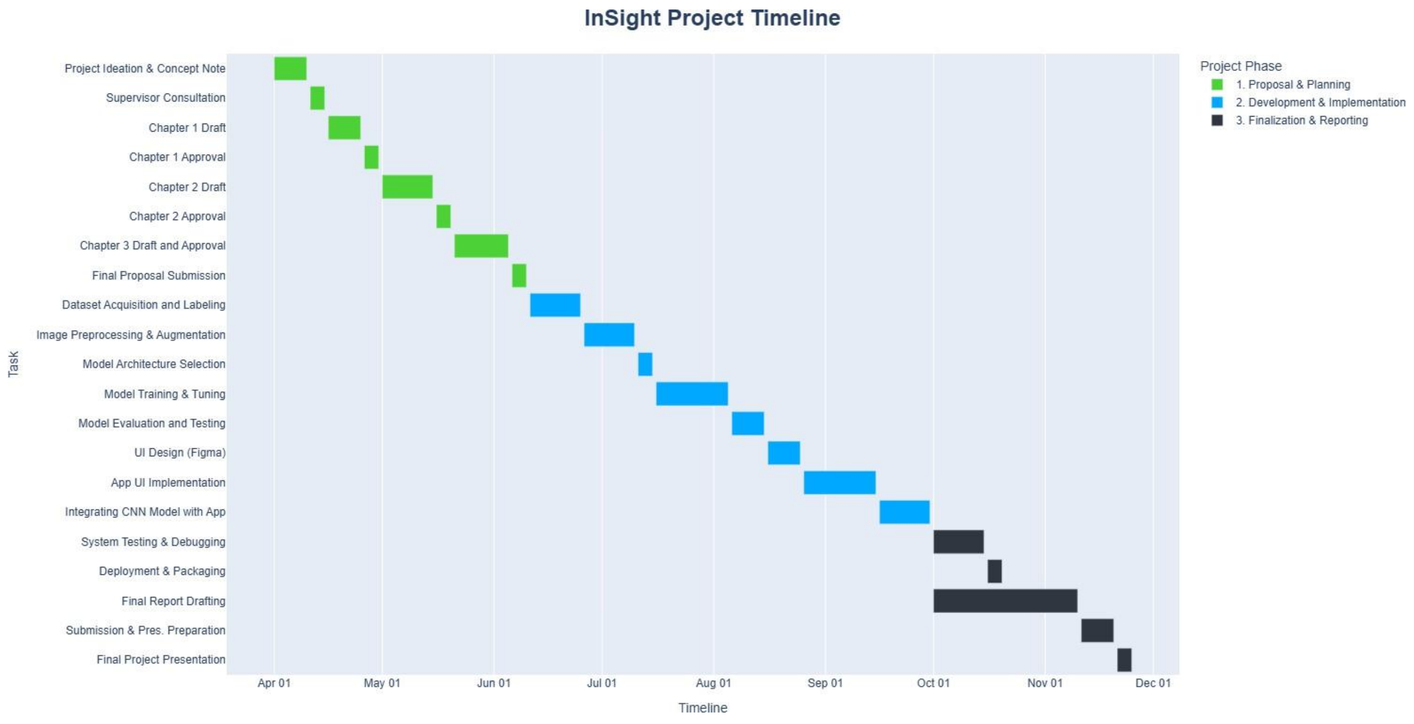
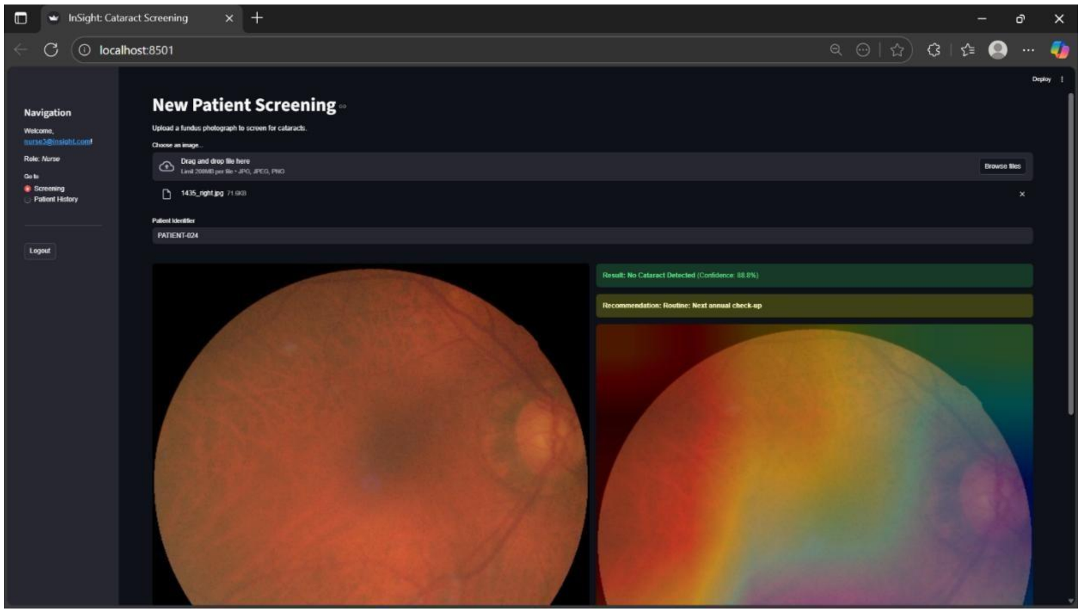


Figure 6.1: Gantt Chart

Appendix 2: InSight Prediction Interface and Sample Ocular Screening Report

This section illustrates the user interface of the InSight web application, demonstrating the workflow from image upload to CNN-based prediction. It also presents a sample of the automated PDF screening report, showcasing how diagnostic results are formatted and delivered to the user for record-keeping.



New Patient Screening

Upload a fundus photograph to screen for cataracts.

Choose an image...

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

1435_right.jpg 71.9KB

Patient Identifier

PATIENT-024

Result: No Cataract Detected (Confidence: 88.8%)

Recommendation: Routine: Next annual check-up

Figure 6.2: InSight Prediction Interface

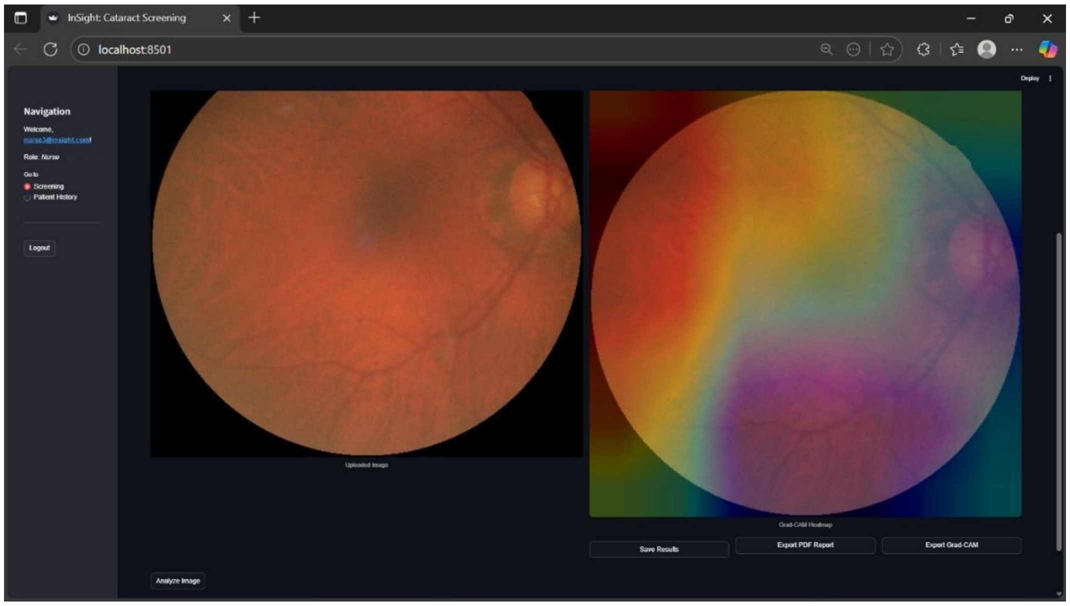
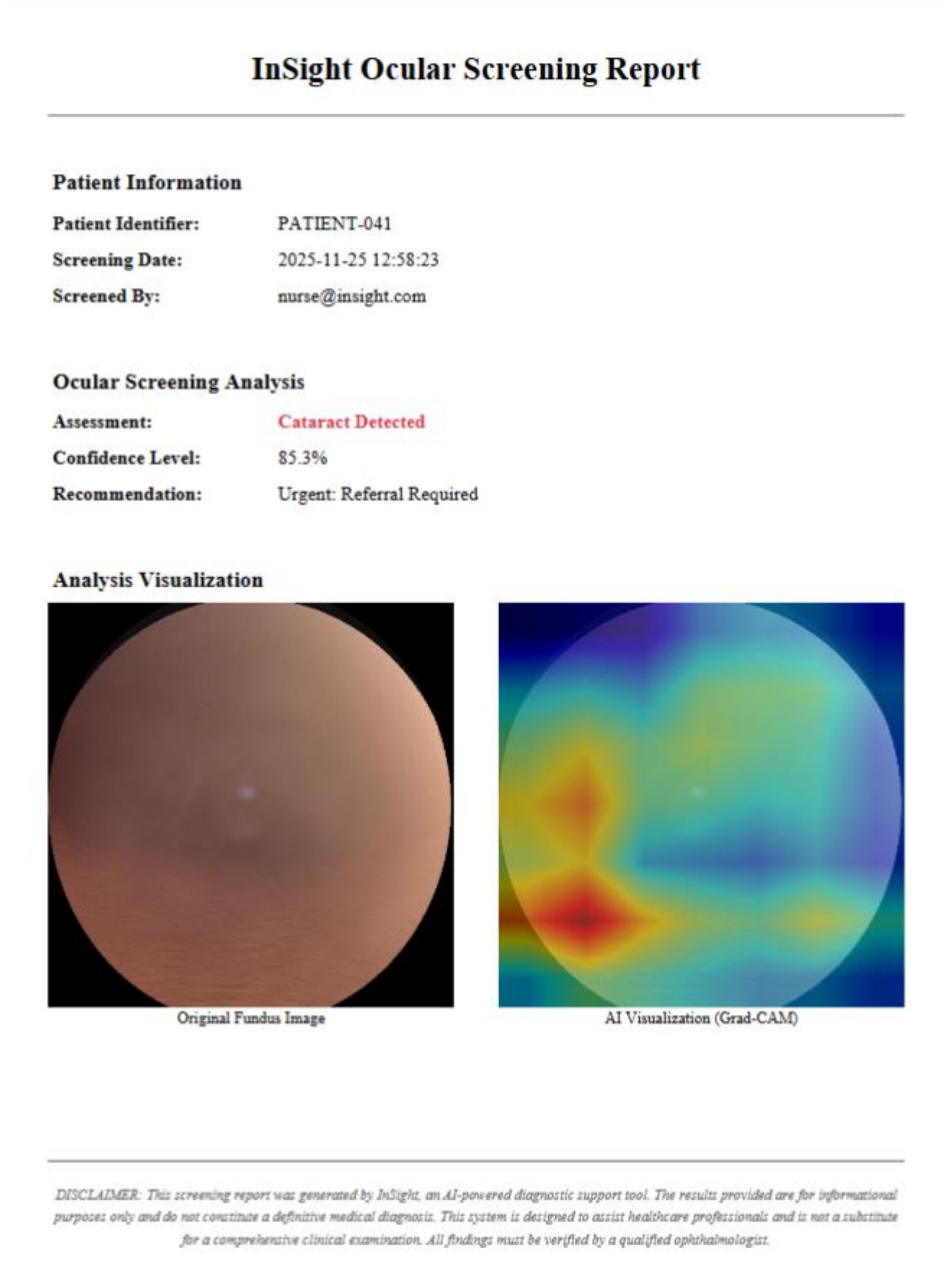


Figure 6.3: InSight Prediction Interface with GradCAM Overlay



InSight Ocular Screening Report

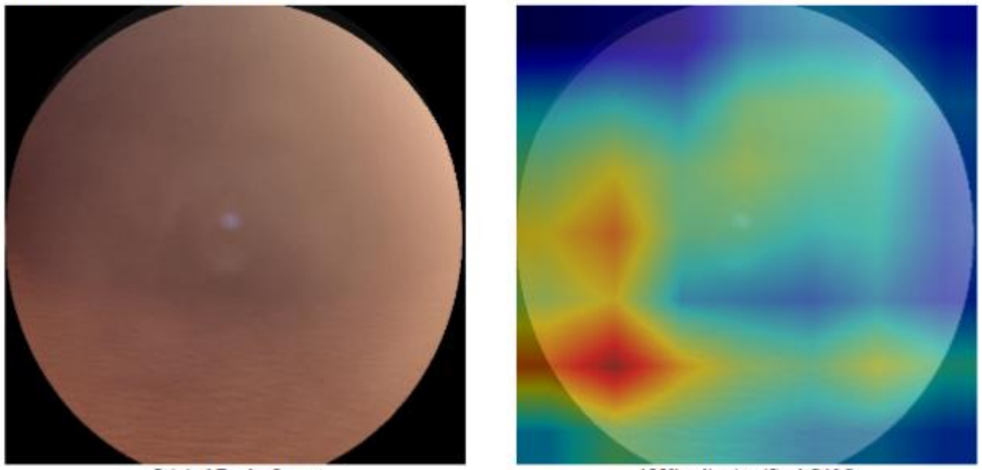
Patient Information

Patient Identifier:	PATIENT-041
Screening Date:	2025-11-25 12:58:23
Screened By:	nurse@insight.com

Ocular Screening Analysis

Assessment:	Cataract Detected
Confidence Level:	85.3%
Recommendation:	Urgent: Referral Required

Analysis Visualization



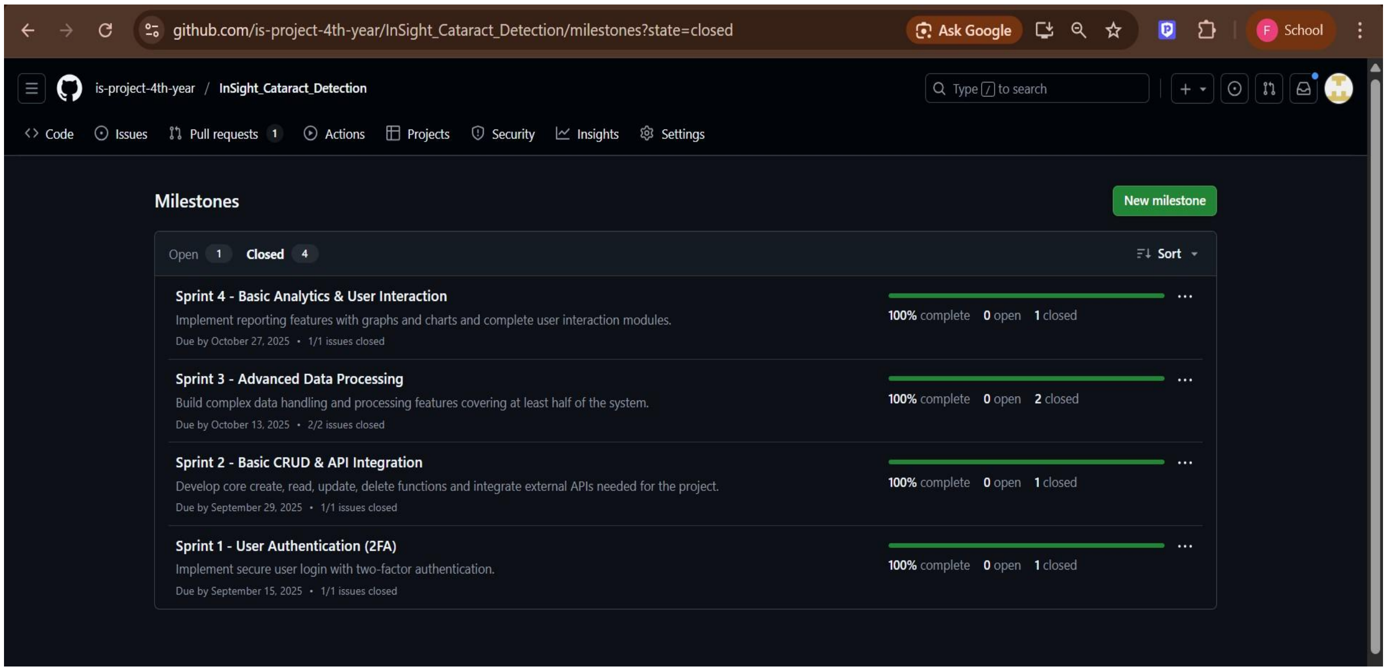
DISCLAIMER: This screening report was generated by InSight, an AI-powered diagnostic support tool. The results provided are for informational purposes only and do not constitute a definitive medical diagnosis. This system is designed to assist healthcare professionals and is not a substitute for a comprehensive clinical examination. All findings must be verified by a qualified ophthalmologist.

Figure 6.4: Sample Screening Report

Appendix 3: Project Repository and Development Milestones

The complete source code, revision history, and issue tracking for the InSight application are hosted on GitHub. The repository serves as the central version control system for the project.

Repository URL: https://github.com/is-project-4th-year/InSight_Cataract_Detection/tree/main



Sprint 4 - Basic Analytics & User Interaction
Implement reporting features with graphs and charts and complete user interaction modules.
Due by October 27, 2025 • 1/1 issues closed
100% complete 0 open 1 closed

Sprint 3 - Advanced Data Processing
Build complex data handling and processing features covering at least half of the system.
Due by October 13, 2025 • 2/2 issues closed
100% complete 0 open 2 closed

Sprint 2 - Basic CRUD & API Integration
Develop core create, read, update, delete functions and integrate external APIs needed for the project.
Due by September 29, 2025 • 1/1 issues closed
100% complete 0 open 1 closed

Sprint 1 - User Authentication (2FA)
Implement secure user login with two-factor authentication.
Due by September 15, 2025 • 1/1 issues closed
100% complete 0 open 1 closed

Figure 6.5: GitHub Milestones view demonstrating sprint planning