PROJECT REPORT

ON

DIABETIC RETINOPATHY DETECTION SYSTEM

Using Deep Learning (CNN)

Keywords: Diabetic Retinopathy, Convolutional Neural Network, Deep Learning, Image Classification, Flask, TensorFlow, Keras, Medical Imaging, Binary Classification

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Project Overview

Diabetic Retinopathy (DR) is a serious complication of diabetes mellitus that damages the blood vessels of the retina, leading to progressive vision loss and eventual blindness if not detected and treated in time. It is currently the leading cause of blindness among working-age adults worldwide. Millions of people suffer from this condition, and the problem is particularly severe in rural and remote areas where access to trained ophthalmologists is highly limited.

The Aravind Eye Hospital in India represents a classic example of this challenge — technicians travel to rural communities to capture retinal fundus images using specialized cameras, and these images are then sent to specialist physicians in hospitals for manual review and diagnosis. This process is inherently slow, expensive, inconsistent due to physician fatigue, and practically impossible to scale to the growing global diabetic population.

The project titled "Diabetic Retinopathy Detection System" aims to solve this problem by applying Artificial Intelligence and Deep Learning techniques. The system automatically classifies retinal fundus images as Infected (signs of DR present) or Uninfected (no DR signs) using a trained Convolutional Neural Network (CNN). The model is integrated into a Flask-based web application, allowing community health workers and clinicians to upload patient retinal photographs and receive an instant AI-generated screening result.

This project demonstrates the practical application of Machine Learning in medical image analysis and healthcare automation, providing a smart and scalable solution to support early detection and ultimately reduce the burden of preventable blindness caused by diabetic retinopathy.

1.2 Purpose

The main purpose of this project is to develop an automated detection system that identifies signs of diabetic retinopathy in retinal fundus images using deep learning-based image classification.

The specific objectives are:

- To reduce delays in diabetic retinopathy screening by automating the image analysis process.
- To improve diagnostic consistency and reduce human error in retinal image interpretation.
- To provide accessible screening tools for healthcare workers in rural and resource-limited settings.

- To implement a practical deep learning pipeline including data preprocessing, model training, and web deployment.
- To demonstrate how AI can scale the diagnostic capacity of specialist physicians beyond geographic limitations.
- To gain hands-on experience in CNN model design, dataset augmentation, backend integration with Flask, and end-to-end web application deployment.

2. IDEATION PHASE

2.1 Problem Statement

The ideation phase involved identifying real-world challenges in the medical field and exploring multiple technological approaches to address them. The primary focus was understanding the root cause of inefficiencies in the manual retinal image inspection process used in diabetic retinopathy screening programs.

Extensive research was conducted to evaluate how artificial intelligence can assist in visual medical diagnosis. The key problem identified was that manual inspection of retinal fundus photographs by specialist ophthalmologists is extremely time-consuming, expensive, geographically restricted, and prone to fatigue-induced inconsistency.

User-centric analysis played a major role in shaping the solution. Diabetic patients in rural areas expressed concerns about the unavailability of eye specialists near their communities. Healthcare workers highlighted the impracticality of transporting large volumes of retinal images for expert review. Hospital administrators required a system that could deliver reliable screening results quickly without demanding advanced technical skills from frontline workers.

Various solution strategies were discussed during the ideation phase, including traditional computer vision techniques and classical machine learning approaches. However, deep learning-based Convolutional Neural Networks were selected due to their superior ability to automatically extract complex hierarchical features from medical images without requiring manual feature engineering.

Prototyping ideas included designing an intuitive web-based upload interface, ensuring real-time processing of images on standard hardware, and maintaining scalability for future expansion to multi-class severity grading. The ideation process ensured that the final solution was both technically feasible and practically applicable in real-world healthcare environments.

2.2 Empathy Map Canvas

Users:

- Diabetic patients in rural and urban communities
- Community health workers and field technicians
- Ophthalmologists and specialist physicians
- Hospital administrators and NGO healthcare program managers

What Users Think:

- Travelling to a specialist for a routine eye check-up is difficult and costly.

- Manual review of retinal images takes too long when dealing with thousands of patients.
- There must be a smarter way to screen for retinopathy without requiring a doctor in every village.

What Users Feel:

- Frustrated by the lack of accessible eye care in their communities.
- Anxious about vision loss and the consequences of late diagnosis.
- Overwhelmed by the volume of patients to screen manually.
- Hopeful that technology can improve the quality and reach of healthcare.

What Users Need:

- A fast and reliable automated screening system.
- Accurate binary classification of retinal images.
- An easy-to-use application that does not require medical expertise to operate.
- A cost-effective solution deployable in low-resource settings.

Understanding these user needs helped in designing a simple web-based interface with fast prediction capability, clear result display, and clinical guidance for follow-up action.

2.3 Brainstorming

During brainstorming, multiple technical approaches were considered for solving the retinal image classification problem:

- Using traditional image processing techniques such as color thresholding, texture feature extraction, and edge detection to identify lesions.
- Using classical machine learning algorithms such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Random Forest classifiers on manually extracted features.
- Using Deep Learning models, specifically Convolutional Neural Networks (CNN), which can automatically learn visual patterns from raw image data.
- Building a desktop application for offline use in clinical settings.
- Creating a web-based application for broad accessibility across devices and locations.

After evaluating each approach on the basis of accuracy, scalability, implementation complexity, and deployability, CNN-based Deep Learning was selected as the final approach. CNNs provide superior performance in image classification tasks because they automatically learn hierarchical spatial features, handle image variation robustly, and generalize well when trained with data augmentation. A Flask web application was chosen for deployment due to its lightweight nature and ease of integration with Python-based machine learning models.

3. REQUIREMENT ANALYSIS

Requirement analysis was conducted systematically to ensure the solution meets both user expectations and technical feasibility standards. Both functional and non-functional requirements were clearly defined based on user interviews, literature review, and technical exploration.

3.1 Customer Journey Map

1. The user (health worker or doctor) opens the web application in a browser.
2. The user navigates to the upload section of the home page.
3. The user selects or drags-and-drops a retinal fundus image (JPG, PNG, or TIFF format).
4. A preview of the uploaded image is displayed on the screen.
5. The user clicks the Analyze Image button to submit the image.
6. The system preprocesses the image: resizing to 64×64 pixels and normalizing pixel values.
7. The trained CNN model analyzes the image and generates a prediction.
8. The system classifies the image as Infected (DR present) or Uninfected (no DR).
9. The result is displayed with a confidence percentage and clinical guidance message.
10. The user can print the result or analyze another image.

This journey ensures a smooth, simple, and clinically meaningful user experience with no technical expertise required from the end user.

3.2 Solution Requirement

Functional Requirements:

- Image upload feature with support for JPG, PNG, TIFF, and GIF formats.
- Drag-and-drop interface with real-time image preview before submission.
- Image preprocessing module: resize to 64×64 pixels, normalize pixel values to [0,1].
- TIFF-to-JPEG automatic conversion for browser display compatibility.
- CNN-based binary classification model (Infected / Uninfected).
- Display prediction result with label, confidence score, and clinical guidance.
- Error handling for invalid or missing input files.
- Loading overlay animation to indicate processing in progress.

Functional requirements ensure the system can accept retinal images, preprocess them correctly, run model inference, and present results clearly. Backend integration allows smooth communication between the web interface and the machine learning model.

Non-Functional Requirements:

- High prediction accuracy (target: ≥70% on validation data).
- Fast response time: prediction result delivered within 2–5 seconds.
- User-friendly, responsive interface accessible on desktop and mobile browsers.
- Secure image processing: images processed locally without external data transmission.
- Scalability: architecture supports model replacement with more advanced CNN without redesign.
- Reliability: system gracefully handles incorrect input formats and edge cases.

Non-functional requirements emphasized performance efficiency, security, usability, and future scalability. Data privacy is maintained by processing all images locally on the server without cloud transmission.

3.3 Data Flow Diagram

Level 0 DFD (Context Diagram):

User → Upload Retinal Image → Diabetic Retinopathy Detection System → Prediction Result → User

Level 1 DFD:

11. Image Input (User uploads retinal fundus photograph)
12. Image Preprocessing (Resize to 64×64, normalize pixels, TIFF conversion if required)
13. Feature Extraction (CNN convolutional layers extract spatial features)
14. Model Classification (Fully connected layers + sigmoid activation produce binary prediction)
15. Output Generation (Prediction label + confidence score rendered to user)

The data flow architecture ensures proper information processing at each stage. The structured DFD levels describe how image input flows through preprocessing, feature extraction, classification, and result generation modules. This structured approach ensures maintainability and ease of debugging.

3.4 Technology Stack

Frontend:

- HTML5 (Structure and layout)
- CSS3 (Styling and responsive design)
- JavaScript ES6 (Drag-drop, image preview, form handling)
- Bootstrap 5 (Responsive grid and UI components — via CDN)
- Font Awesome 6 (Icon library — via CDN)

Backend:

- Python 3.8 (Core programming language)
- Flask 1.1.2 (Lightweight web framework for API integration)
- Pillow / PIL (Image format conversion and TIFF support)

Machine Learning:

- TensorFlow 2.3.1 (Deep learning framework)
- Keras 2.4.3 (High-level neural network API)
- OpenCV 4.4.0 (Image preprocessing and color space operations)
- NumPy 1.18.5 (Array operations and data manipulation)

Dataset:

- Labeled retinal fundus images classified as Infected (DR present) and Uninfected (healthy retina)
- Training directory: output/train/ | Validation directory: output/val/

Tools:

- VS Code (Development environment)
- Jupyter Notebook (Model training and experimentation)
- GitHub (Version control and project repository)

4. PROJECT DESIGN PHASE

4.1 Problem Solution Fit

Problem	Solution
Manual retinal image review is slow and expensive.	CNN model delivers automated prediction in under 5 seconds.
Specialist ophthalmologists are unavailable in rural areas.	Web-based tool allows any health worker to upload and analyze images.
Manual inspection is prone to fatigue and human error.	AI model delivers consistent, reproducible predictions every time.
TIFF format images from medical cameras are not browser-compatible.	Automatic TIFF-to-JPEG conversion handled server-side by Pillow.
Screening programs cannot scale to millions of patients.	Web application architecture supports multiple concurrent users.

The proposed solution directly addresses each inefficiency of manual inspection by providing instant, accurate, and accessible automated detection through an AI-powered web application.

4.2 Proposed Solution

The proposed system uses a trained CNN model integrated into a Flask web application. The complete working process is as follows:

16. User opens the web application and uploads a retinal fundus image via the upload interface.
17. The image is saved to the server's uploads directory.
18. If the uploaded image is in TIFF format, it is automatically converted to JPEG using Pillow for browser display compatibility.
19. The image is resized to 64×64 pixels to match the CNN input dimensions.
20. Pixel values are normalized by dividing by 255 to bring them into the range [0.0, 1.0].
21. The normalized image array is passed through the trained CNN model using model.predict().
22. The CNN's convolutional layers extract spatial feature maps from the retinal image.
23. The fully connected layers and sigmoid output neuron produce a binary probability score.
24. The argmax of the prediction maps to the class label: 0 = Infected, 1 = Uninfected.

25. The prediction label and confidence percentage are displayed to the user with clinical guidance.

The system is designed to be simple for end users while being technically robust. The modular design allows the CNN model to be upgraded independently without redesigning the web application.

4.3 Solution Architecture

Architecture Components:

- User Interface Layer — HTML5 templates rendered by Jinja2; Bootstrap 5 responsive design; JavaScript for drag-drop and image preview.
- Application Layer — Flask web server handling HTTP routing, file I/O, image preprocessing, and template rendering.
- Machine Learning Model Layer — Pre-trained Keras CNN model (model.h5) loaded at startup; performs binary image classification.
- Dataset Storage — Labeled retinal fundus images organized in class subdirectories for training and validation.
- Prediction Engine — api() function that loads, resizes, normalizes, and classifies uploaded images using the trained model.

Architecture Flow:

User → Web Interface (index.html) → Flask Backend (app.py) → CNN Model (model.h5) → Prediction Result → Web Interface (predict.html) → User

Layer	Technology	Responsibility
Presentation	HTML5, CSS3, Bootstrap 5, JavaScript	User interface, image upload, result display
Application	Flask (Python 3.8), Jinja2, Pillow	Request routing, file handling, preprocessing
Model	TensorFlow, Keras, CNN (model.h5)	Image classification, confidence scoring

This modular three-tier architecture allows easy maintenance, testing, and future upgrades. Each layer communicates through well-defined interfaces, ensuring that changes to one component do not break others.

5. PROJECT PLANNING PHASE

5.1 Project Planning

Development Phases:

- Phase 1: Requirement Analysis — Define problem, study existing solutions, gather user requirements.
- Phase 2: Dataset Collection and Cleaning — Source labeled retinal fundus image dataset; remove duplicates and corrupted images; organize into train/val directories.
- Phase 3: Data Preprocessing — Implement brightness adjustment, rescaling, augmentation pipeline using Keras ImageDataGenerator and OpenCV.
- Phase 4: Model Design and Training — Build Sequential CNN architecture; compile with Adam optimizer; train for 10 epochs with batch size 32.
- Phase 5: Model Evaluation — Assess accuracy, loss on validation set; review misclassified samples; save model as model.h5.
- Phase 6: Integration with Web Application — Build Flask backend; create HTML templates; integrate model.h5 for inference.
- Phase 7: Testing and Debugging — Functional testing of all routes; test multiple image formats; validate error handling.
- Phase 8: Deployment and Documentation — Deploy locally; prepare project report and technical documentation.

Total Duration: Approximately 4–6 weeks.

Phase	Activity	Duration	Output
1	Requirement Analysis	Week 1	Requirements document
2	Dataset Collection & Cleaning	Week 1–2	Organized dataset
3	Data Preprocessing & Augmentation	Week 2	Preprocessing pipeline
4	Model Design and Training	Week 2–3	Trained model.h5
5	Model Evaluation	Week 3	Accuracy metrics
6	Web Application Integration	Week 4	Working web app
7	Testing and Debugging	Week 5	Test report
8	Deployment and Documentation	Week 5–6	Final project report

Project planning was structured into multiple well-defined phases to ensure systematic development. Dataset collection required careful selection of high-quality labeled retinal images representing both infected and uninfected conditions. Data cleaning involved removing corrupted or duplicate images and organizing them into the required directory structure.

Model development included selecting an appropriate CNN architecture, tuning hyperparameters, and implementing data augmentation to prevent overfitting. Integration planning connected the Flask backend with the HTML frontend templates. Proper documentation was maintained at every stage to support future enhancements.

Risk management strategies were implemented to handle potential technical challenges including GPU memory limitations during training, dataset class imbalance, and TIFF format compatibility issues in the web application.

6. PROJECT DEVELOPMENT PHASE

6.1 CNN Model Architecture

The Convolutional Neural Network was built using the Keras Sequential API with the following layer configuration:

#	Layer	Configuration	Output Shape	Activation
1	**Conv2D**	32 filters, 3×3 kernel	(62, 62, 32)	ReLU
2	**MaxPooling2D**	Pool size: 2×2	(31, 31, 32)	—
3	**Conv2D**	64 filters, 3×3 kernel	(29, 29, 64)	ReLU
4	**MaxPooling2D**	Pool size: 2×2	(14, 14, 64)	—
5	**Flatten**	—	(12544)	—
6	**Dense (Hidden)**	128 neurons	(128)	ReLU
7	**Dense (Output)**	1 neuron — Binary	(1)	Sigmoid

Optimizer: Adam (Adaptive Moment Estimation)

Loss Function: Binary Cross-Entropy

Metric: Accuracy

Epochs: 10

Batch Size: 32

Input Shape: 64 × 64 × 3 (RGB images)

Output: Binary — 0: Infected (DR present), 1: Uninfected (Healthy retina)

6.2 Data Preprocessing & Augmentation

A custom preprocessing function brightness_adjustment() was implemented using OpenCV to simulate real-world lighting variations in retinal photographs. The function converts images from RGB to HSV color space and back to BGR, normalizing image brightness across different camera settings.

The following augmentation techniques were applied using Keras ImageDataGenerator during training:

- Rescaling: Pixel values divided by 255 to normalize to the range [0.0, 1.0].
- Brightness Adjustment: Custom HSV-based brightness normalization applied to every image.
- Shear Transformation: Random shear range of 0.2 to simulate camera angle variations.
- Zoom: Random zoom range of 0.2 to account for different focal distances.
- Horizontal Flip: Random left-right mirroring to double effective training data and handle left/right eye symmetry.

These augmentation techniques significantly improve model generalization by exposing the CNN to diverse image conditions during training, reducing overfitting and improving performance on unseen real-world retinal photographs.

6.3 Web Application Development

The web application was developed using Flask (Python) as the backend framework. The application consists of the following key components:

Backend (app.py):

- Flask routes: / (home page), /upload (image submission and prediction), /uploads/<filename> (image serving).
- Model loaded once at startup using tensorflow.keras.models.load_model('model.h5').
- api() function handles: image loading, dimension expansion, normalization, and model.predict() inference.
- Automatic TIFF to JPEG conversion using Pillow for images captured by medical cameras.
- GPU auto-detection: uses GPU if available, gracefully falls back to CPU.

Frontend (index.html & predict.html):

- Home page features drag-and-drop upload area, real-time image preview using FileReader API, and loading overlay animation.
- Results page displays uploaded image, color-coded prediction label (red: Infected, green: Uninfected), animated confidence bar, and clinical guidance.
- Bootstrap 5 provides responsive layout for both desktop and mobile browsers.
- Jinja2 templating renders dynamic prediction results passed from Flask as template variables.

7. PROJECT DOCUMENTATION & TESTING

7.1 Performance Testing

The model was evaluated on the validation dataset after training for 10 epochs. The following performance metrics were recorded:

Metrics Used:

- Accuracy — Percentage of correctly classified images over total test samples.
- Loss — Binary cross-entropy loss value indicating how well the model fits the training data.
- Precision — Of all images predicted as Infected, how many actually have DR.
- Recall — Of all actual DR cases, how many were correctly identified by the model.

Performance Results:

Metric	Value	Remarks
Training Accuracy	~80–85%	Consistent improvement over 10 epochs
Validation Accuracy	~73%	Final model accuracy on unseen data
Training Loss	~0.35	Binary cross-entropy, decreasing trend
Validation Loss	~0.45	Acceptable generalization gap
Response Time	< 5 seconds	Per single image prediction (CPU mode)
Supported Formats	JPG, PNG, TIFF, GIF	All standard medical image formats

Functional and performance testing ensured the reliability and robustness of the developed system. Unit testing was conducted for each module including image preprocessing, prediction API endpoint, TIFF conversion, and result rendering.

Performance testing included measuring response time under different image format inputs. Error handling was verified by submitting invalid file types and empty form submissions. The system consistently returned informative error messages and never crashed under invalid input.

The system successfully classified retinal images of varying quality and lighting conditions with high reliability. These results confirm the practical feasibility of the developed solution as a screening assistance tool.

Functional Test Cases:

No.	Test Case	Expected Result	Status
1	Upload JPG retinal image	Prediction + confidence displayed	PASS
2	Upload PNG retinal image	Prediction + confidence displayed	PASS
3	Upload TIFF retinal image	Auto-converts to JPEG; prediction shown	PASS
4	Submit without selecting a file	Flash error message displayed	PASS
5	Drag and drop file to upload area	File selected; preview shown	PASS
6	Analyze Another Image button	Redirects to home page	PASS
7	Print Results button	Browser print dialog opens	PASS
8	Mobile responsive layout	Layout adapts to small screen	PASS

8. RESULTS

8.1 Output Screenshots

The system provides the following user-facing pages and outputs:

- Home Page — Gradient-styled upload interface with drag-and-drop functionality.
- Image Preview Panel — Real-time display of selected retinal image before submission.
- Loading Screen — Animated spinner overlay displayed during AI analysis.
- Prediction Result Page — Shows uploaded image, diagnosis label, confidence bar, and clinical guidance.
- Print Results — Browser print-ready result page for clinical record keeping.

Sample Outputs:

Input: Retinal fundus image of a diabetic patient with visible hemorrhages
Output: "Infected" — Signs of Diabetic Retinopathy Detected (Confidence: 87.45%)
Guidance: Medical consultation recommended

Input: Retinal fundus image of a healthy eye with no visible lesions
Output: "Uninfected" — No Signs of Diabetic Retinopathy (Confidence: 82.13%)
Guidance: Regular monitoring advised

The model correctly identifies the majority of test samples across different image formats, lighting conditions, and retinal camera types. Results are consistent and reproducible for the same input image.

Home Page Features:

- Purple gradient background with a frosted glass upload card.
- Drag-and-drop upload area with visual hover effect.
- Image preview displayed below the upload area.
- Three feature cards: Deep Learning (73% Accuracy), Fast Results, Secure Processing.
- Analyze Image button (enabled only after file selection).

Result Page Features:

- Uploaded retinal image shown prominently.
- Color-coded diagnosis: Red for Infected, Green for Uninfected.
- Animated confidence progress bar with percentage display.
- Clinical guidance message and severity indicator badge.
- Option to Analyze Another Image or Print Results.

9. ADVANTAGES & DISADVANTAGES

The system offers numerous advantages including full automation of retinal image analysis, significant reduction of manual ophthalmologist workload, consistent and reproducible predictions, and scalability to process large volumes of patient images rapidly. It enables early detection of diabetic retinopathy in communities where specialist eye care is inaccessible, potentially preventing thousands of cases of preventable blindness.

However, certain limitations exist. The system performs binary classification only and does not grade the severity of retinopathy (mild, moderate, severe, proliferative). The current model accuracy of 73% is suitable for screening assistance but falls below the clinical deployment threshold typically required for standalone diagnostic tools. Image quality variations can affect prediction confidence.

Despite these challenges, the system remains highly beneficial in practical screening applications. Continuous dataset expansion, retraining with deeper architectures, and integration with transfer learning models can further enhance accuracy and clinical reliability.

Advantages:

- Automates diabetic retinopathy screening, saving time and reducing specialist workload.
- Provides fast prediction results within seconds compared to days for manual review.
- Delivers consistent, fatigue-free classification for every submitted image.
- Accessible via standard web browser — no software installation required for end users.
- Supports multiple medical image formats including TIFF used in ophthalmological equipment.
- Easily deployable on standard hardware without specialized GPU requirements for inference.
- Can be expanded to multi-class severity grading with additional training data.

Disadvantages:

- Binary classification only — does not differentiate between severity levels of DR.
- Current 73% accuracy is insufficient for autonomous clinical diagnosis without specialist oversight.
- Model trained on 64×64 resolution images — fine retinal lesion details may be lost.
- Performance depends on input image quality — low-contrast or blurry images reduce accuracy.
- Cannot detect internal ocular complications or systemic diabetic changes.

- Requires retraining when expanding to new retinal conditions or additional disease categories.
- No user authentication or encrypted image storage in the current implementation.

10. CONCLUSION

The Diabetic Retinopathy Detection System successfully demonstrates the use of deep learning in automated medical image analysis for early eye disease screening. The system achieves approximately 73% classification accuracy and provides an efficient, accessible alternative to manual retinal image inspection by specialist ophthalmologists.

In conclusion, this project successfully demonstrates the practical application of Artificial Intelligence in healthcare quality management. The integration of CNN-based deep learning with Flask web technologies creates a scalable and efficient retinal image screening tool that can operate without specialist expertise from the end user.

The project enhances understanding of the complete machine learning workflow including dataset preparation and augmentation, model design and training, evaluation metrics analysis, backend API integration, frontend development, and web deployment. It also emphasizes the importance of user-centric design in creating technology solutions for real-world healthcare challenges.

The system contributes toward the global goal of reducing preventable blindness caused by diabetic retinopathy by providing fast, consistent, and accessible screening. With further enhancements including transfer learning, higher resolution inputs, and cloud deployment, this system has strong potential for commercial and NGO-level healthcare implementation worldwide.

11. FUTURE SCOPE

- Transfer Learning Integration — Fine-tune pre-trained models such as ResNet50, EfficientNetB0, or InceptionV3 on retinal image data to achieve classification accuracy above 90% and match clinical standards.
- Multi-class Severity Grading — Extend the model to classify retinopathy into five severity levels: No DR, Mild, Moderate, Severe, and Proliferative DR, enabling more clinically actionable outputs.
- Higher Input Resolution — Increase input image size to 224×224 or 512×512 pixels to preserve fine retinal structures such as microaneurysms, dot hemorrhages, and exudates critical for early-stage detection.
- Grad-CAM Visualization — Implement Gradient-weighted Class Activation Mapping to overlay heatmaps on retinal images, showing which regions of the retina influenced the model's prediction for explainability.
- Mobile Application Development — Build a native Android/iOS application enabling field health workers to capture and analyze retinal images directly using the device camera.
- Real-time Camera Detection — Integrate with retinal fundus cameras or smartphone adapters for real-time live image capture and instant analysis.
- Cloud Deployment — Containerize the application using Docker and deploy on AWS, Google Cloud Platform, or Azure for internet-accessible deployment and centralized monitoring.
- REST API Endpoint — Expose a standardized REST API for integration with hospital management systems, electronic health record (EHR) platforms, and telemedicine applications.
- User Authentication and Patient History — Implement patient login system with encrypted image storage and historical screening record tracking for longitudinal disease monitoring.
- Integration with IoT Devices — Connect with IoT-enabled ophthalmic equipment in automated screening kiosks deployed in pharmacies, rural health centers, and diabetes clinics.
- Internal Defect Detection — Explore hyperspectral imaging and advanced deep learning techniques for detecting internal ocular complications not visible on standard fundus photographs.

12. APPENDIX

Source Code:

https://github.com/uday1874533/DiabeticRetinopathyDetection

GitHub Repository:

https://github.com/uday1874533/DiabeticRetinopathyDetection

Project Demo Video:

https://drive.google.com/file/d/15BPmk42279u1QbG7P8QOUoL3jTDoddDg/view?usp=drivesdk]

Dataset:

Labeled retinal fundus images — Infected (Diabetic Retinopathy) and Uninfected (Healthy Retina). Dataset organized as train/, val/, and test/ with class subdirectories.

Model File:

model.h5 — Trained Keras Sequential CNN model. Saved in HDF5 format. Located in Deploy/model.h5. Loaded at Flask application startup for inference.