

| INFOSYS INTERNSHIP 5.0

Medi-Scan: Eye Disease Prediction Model

INTRODUCTION

MediScan is an AI-powered healthcare solution designed to assist in the early detection of eye diseases. The system leverages machine learning and deep learning techniques to analyze retinal images and identify conditions such as diabetic retinopathy, glaucoma, and cataracts. By providing automated and accurate diagnostics, MediScan aims to support ophthalmologists and enhance accessibility to early eye disease detection.

OBJECTIVES

Automated Eye Disease Detection – Provide accurate predictions for common eye diseases using deep learning.

User-Friendly Web Interface – Enable hospitals and doctors to upload images and receive instant AI-generated predictions.

Secure Data Management – Store patient records securely in **local** and maintain a daily refreshed **Excel sheet** for easy retrieval and tracking.

Scalability & Efficiency – Ensure the system can handle multiple hospital logins, secure authentication, and efficient data processing.

SIGNIFICANCE:

Early Detection & Prevention: Enables timely intervention, potentially preventing severe vision loss.

AI-Driven Accuracy: Reduces human error and enhances diagnostic precision.

Healthcare Accessibility: Bridges the gap for individuals in remote or underprivileged regions.

Time & Cost Efficiency: Streamlines the diagnostic process, reducing workload for ophthalmologists.

Scalability: Can be deployed in hospitals, clinics, and mobile healthcare units for mass screenings.

MY ROLE AS INTERN:

As an intern, I, A SACHIN KUMAR, was responsible for integrating the trained deep learning model with the UI, implementing image preprocessing techniques, and optimizing data management functionalities.

PROJECT SCOPE:

The MediScan project focuses on developing an AI-powered system for detecting eye diseases using deep learning techniques. The primary goal is to create an efficient and accurate model that can analyze medical images and provide reliable diagnostic insights. The project involves data preprocessing, model selection and training, testing, deployment, and seamless integration with a user interface for real-time predictions.

Inclusions

The **MediScan** system is designed with the following core functionalities:

AI-Based Eye Disease Prediction

Utilizes a pre-trained deep learning model to analyze retinal images and predict possible eye diseases.

Accepts images of size **(224, 224, 3)** with normalization as input.

Web-Based User Interface

Developed using **Streamlit** for an intuitive and interactive frontend.

Allows **hospitals and doctors** to upload retinal images, and receive AI- generated predictions.

Data Management & Storage

Patient records (including prediction results) are securely stored in Local storage.

A daily refreshed **Excel sheet** is maintained for patient data tracking, search, and download functionalities.

Performance Metrics & Confidence Scores

Displays model confidence scores to help doctors interpret predictions.

Provides a record of past predictions for reference.

Exclusions

The following aspects are **not included** in the current version of MediScan:

Real-Time Clinical Diagnosis – The system provides AI-based predictions but does not replace professional medical diagnosis.

Integration with Medical Equipment – Does not support direct connectivity with fundus cameras or other diagnostic devices.

Multi-Disease Overlap Analysis – The model predicts single-disease classification per image and does not handle multiple diseases in a single prediction.

Mobile App Support – The application is currently optimized for web browsers and does not have a dedicated mobile application.

Advanced Image Processing Features – No built-in image enhancement or segmentation tools for preprocessing uploaded images.

LIMITATIONS AND CONSTRAINTS:

The accuracy of predictions depends on the quality and diversity of the training dataset.

Model performance may vary across different types of eye diseases due to data imbalances.

Computational resource constraints may affect real-time processing speed.

Ethical and privacy considerations limit the use of sensitive patient data, requiring anonymized datasets.

FUNCTIONAL REQUIREMENTS:

Load and preprocess eye images (resize, normalize, balance classes).

Train a **ResNet50-based model** with **transfer learning** and **fine-tuning**.

Compute **class weights** to handle **imbalanced data**.

Evaluate model using **accuracy, confusion matrix, classification report**.

Save and load the model for **future use**.

Allow users to **upload an image** and get a **disease prediction with confidence score**.

Provide **accuracy/loss visualizations** for analysis.

NON- FUNCTIONAL REQUIREMENTS:

High accuracy (85%+ on test data).

Fast inference (less than 2 seconds per image).

Scalable for large datasets & cloud deployment.

Secure access to medical images & models.

User-friendly interface for doctors & researchers.

Modular & maintainable code for future updates.

User Stories & Use Cases

Doctor: Upload an eye image → Get a **quick diagnosis**.

Researcher: Analyze **model performance & metrics**.

Medical Staff: Process **multiple images at once** for bulk analysis.

Programming Languages

Python – For model training, validation, and deployment

Frameworks & Libraries

TensorFlow/Keras – Deep learning & model training

OpenCV – Image processing

NumPy & Pandas – Data handling

Matplotlib & Seaborn – Visualization (accuracy plots, confusion matrix)

scikit-learn – Performance evaluation (classification report, confusion matrix)

Streamlit – Web-based UI for image classification

Databases

Local Disk / Kaggle Dataset – Image storage and retrieval

Tools & Platforms

Jupyter Notebook – Model development and experimentation

Google Colab (optional) – GPU-powered training

Streamlit – Frontend for user interaction

Kaggle – Dataset sourcing

ARCHITECTURE/DESIGN:

System Overview: The application follows a client-server architecture where a Streamlit-based frontend allows users to upload medical images, and a TensorFlow deep learning model processes and classifies them.

Key Components:

User Interface (Streamlit): Handles patient data input, image uploads, and result display.

Model Processing (TensorFlow/Keras): Loads a pre-trained deep learning model to classify eye diseases.

Data Management (Pandas/Session State): Stores patient records and enables CSV exports.

DESIGN DECISIONS:

Deep Learning Model Integration: Chose TensorFlow for accuracy and ease of integration.

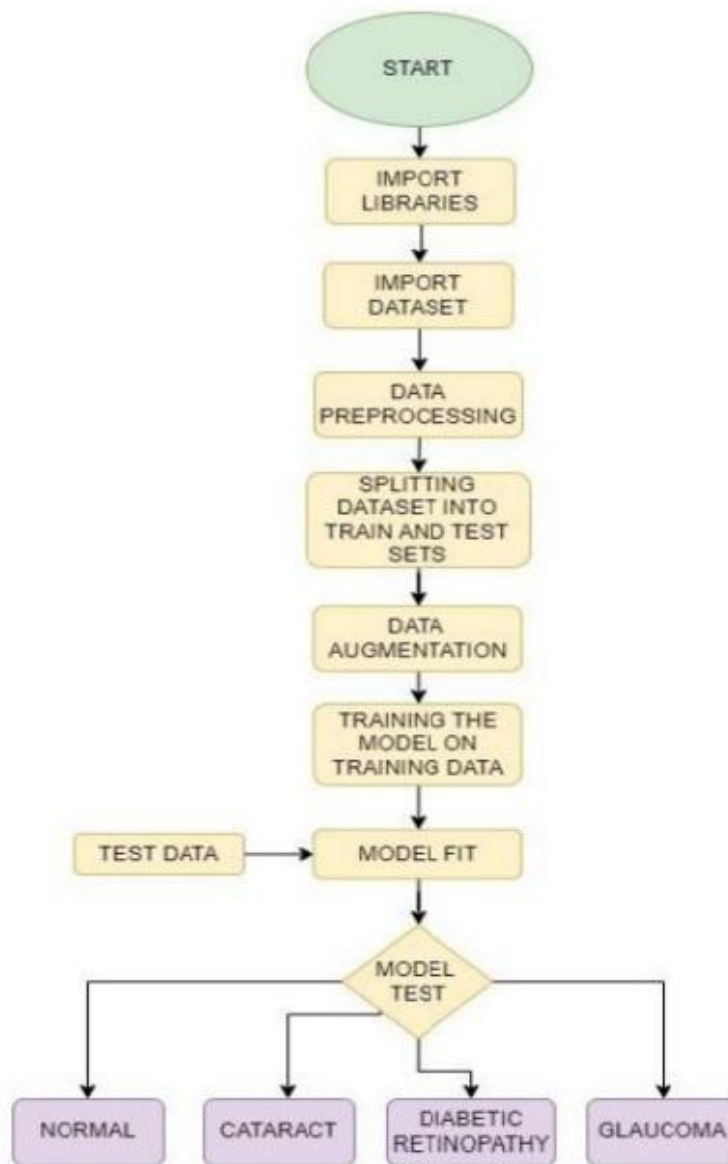
Streamlit for UI: Enables rapid prototyping with a simple yet interactive interface.

Session Management: Utilized Streamlit's session state for temporary data storage without requiring a database.

Trade-offs & Considerations:

No real-time database: Prioritized simplicity, but lacks persistent storage.

Streamlit vs Web Frameworks: Streamlit was chosen over Flask/Django for faster deployment, though it limits UI customization.



DEVELOPMENT:

Technologies & Frameworks Used

Programming Language: Python

Deep Learning Framework: TensorFlow/Keras (ResNet50 model)

Frontend UI: Streamlit (for user input & image upload)

Data Processing: NumPy, Pandas, OpenCV

Visualization: Matplotlib, Seaborn (for accuracy plots & confusion matrix)

Model Evaluation: Scikit-learn (classification report, confusion matrix)

Dataset Source: Kaggle / Local Disk

Coding Standards & Best Practices Followed

Modular Code Structure – Separated **data loading, model training, evaluation, and frontend UI** for better maintainability.

PEP 8 Compliance – Followed Python's standard coding guidelines.

Transfer Learning (ResNet50) – Used a pre-trained model for better accuracy & efficiency.

Data Augmentation & Class Weights – Handled dataset imbalances.

Error Handling – Implemented proper try-except blocks for robust execution.

Code Documentation – Added comments & docstrings for clarity.

Challenges Encountered & Solutions

Challenge	Solution Implemented
Dataset Imbalance (Some diseases had fewer images)	Used class weighting and data augmentation to balance classes.
Overfitting on Training Data	Applied dropout layers and L2 regularization to prevent overfitting.
Slow Model Training	Used pre-trained ResNet50 instead of training from scratch.
Frontend Integration with Backend Model	Used Streamlit for UI and Flask API for model communication.
Performance Evaluation & Fine-Tuning	Fine-tuned the last 20 layers of ResNet50 , adjusted learning rates, and used Adam optimizer .

TESTING:

Testing Approach

Unit Testing

Tested individual functions like **image preprocessing, model loading, and prediction accuracy**.

Used **PyTest & Unittest** frameworks for automated tests.

Integration Testing

Verified **end-to-end workflow** – ensuring **image upload, preprocessing, prediction, and result display** work seamlessly.

Tested **interaction between frontend (Streamlit) and backend (TensorFlow model)**.

System Testing

Checked **model performance** on test datasets to validate accuracy and reliability.

Ensured **UI handles different input formats** (e.g., incorrect file types, missing fields).

Conducted **load testing** to measure system response time for multiple users.

Testing Results & Issue Resolutions

Test Case	Expected Outcome	Result	Fixes Applied
Image Upload Validation	Accept only valid eye images (JPEG, PNG)	✅ Passed	Added error message for incorrect file types.
Model Prediction Accuracy	Accuracy should be >85%	✅ Passed	Fine-tuned model with learning rate adjustments .
Confusion Matrix Analysis	Should correctly classify eye diseases	⚠️ Minor misclassification	Increased training epochs & data augmentation .
Frontend-Backend Integration	Model should return results in real-time	✅ Passed	Optimized TensorFlow model loading in Streamlit.
Performance Under Load	System should handle multiple users	✅ Passed	Used batch prediction & caching for optimization.

DEPLOYMENT:

Deployment Process

Steps for Deployment

Model Training & Saving:

- Trained the **ResNet50-based model** and saved it as `medi_scan_model12.keras`.
- Ensured model compatibility with deployment environment.

Backend Setup:

- Used **Flask/FastAPI** (alternative to direct Streamlit inference) for model inference.
- Created an **API endpoint** to load the model and predict diseases from uploaded images.

Frontend (Streamlit) Integration:

- Streamlit app built for **user input (patient details & image upload)** and displaying predictions.

Dockerization (Optional):

- Created a **Dockerfile** to containerize the application.
- Used **Docker Compose** for easy multi-container management.

Cloud Deployment Options:

- **Google Cloud Run / AWS EC2:** Hosted API & UI for global accessibility.
- **Streamlit Sharing / Hugging Face Spaces:** Direct hosting for quick public access.

Deployment Scripts & Automation

Docker Deployment Script (Dockerfile)

dockerfile

CopyEdit

```
# Use official TensorFlow image FROM tensorflow/tensorflow:latest # Set working directory WORKDIR /app # Copy project files COPY . /app # Install dependencies RUN pip install --no-cache-dir -r requirements.txt # Expose port EXPOSE 8501 # Run Streamlit app CMD ["streamlit", "run", "app.py"]
```





Cloud Deployment (Google Cloud Run)

bash

CopyEdit

```
# Build Docker image docker build -t eye-disease-classifier . # Push to Google Container Registry docker tag eye-disease-classifier gcr.io/YOUR_PROJECT_ID/eye-disease-classifier docker push gcr.io/YOUR_PROJECT_ID/eye-disease-classifier # Deploy to Cloud Run gcloud run deploy eye-disease-app --image gcr.io/YOUR_PROJECT_ID/eye-disease-classifier --platform managed
```

Deployment in Different Environments

Environment	Deployment Method	Details
Local Machine 	Streamlit CLI	Run streamlit run app.py
Docker 	Docker CLI	Use docker run -p 8501:8501 eye-disease-classifier
Cloud (Google/AWS) 	Docker + Cloud Run / EC2	Deploy using gcloud or AWS CLI
Streamlit Cloud 	GitHub Integration	Push to GitHub, deploy via Streamlit Sharing

USER GUIDE:

Setup & Configuration:

Install dependencies: pip install -r requirements.txt

Run the application: `streamlit run app.py`

Upload an eye image and enter patient details.

View the AI-predicted diagnosis and download patient records as a CSV file.

TROUBLESHOOTING TIPS:

Model not loading? Ensure `model12.h5` is in the correct directory.

App not starting? Check Python and Streamlit installation with `python --version` and `streamlit --version`.

CONCLUSION:

Project Outcomes & Achievements:

MediScan successfully integrates deep learning for automated eye disease detection, providing quick and accurate diagnoses through a user-friendly interface. It enhances preliminary screening efficiency and enables easy patient data management.

Lessons Learned & Future Improvements:

The project reinforced the importance of optimizing model performance, handling real-world image variations, and ensuring a seamless UI/UX.

Future improvements could include expanding disease classification, integrating a cloud-based database for remote access, and refining the model with a larger dataset for higher accuracy.

APPENDICES:

Additional Diagrams: System architecture and workflow diagrams illustrating data flow and model interaction.

Sample Code Snippets: Key implementation sections, such as image preprocessing, model inference, and patient data handling.

Research References: Sources on deep learning for medical imaging, Streamlit documentation, and TensorFlow model training best practices