Infosys Internship 5.0

Project Documentation

on

"MediScan: AI-Powered Medical Image Analysis for Disease Diagnosis"

From: **07/10/2024 to 29/11/2024**

Submitted by

Prathmesh Sunil Kadam

Under the Guidance of

Mr. Mohammed Hussain (Project Coordinator)

Mr. Deepak (Mentor)

Acknowledgment

The successful completion of any project is a result of collective efforts, guidance, and support. I would like to express my heartfelt gratitude to all those who contributed to making this project a success.

I am especially thankful to the **Infosys Springboard** team for organizing the internship project titled ***"MediScan: AI-Powered Medical Image Analysis for Disease Diagnosis."*** This initiative provided a wonderful opportunity to work with advanced technologies in medical image analysis, helping me improve my skills and gain valuable knowledge in the field.

I sincerely thank my mentor, **Mr. Deepak**, for his valuable guidance, expertise, and constant support throughout the project. His direction helped me overcome challenges and stay on track. I also extend my gratitude to **Mr. Mohammed Hussain**, our project coordinator, for his support and efficient management, which ensured a smooth and productive workflow.

The scheduled collaboration sessions, held from 7:30 PM to 8:30 PM, were extremely helpful in promoting teamwork, sharing knowledge, and addressing any challenges we faced during the project. These sessions played a key role in maintaining focus and achieving our goals.

I also recognize the importance of the guidelines and milestones discussed during the **Kick-Off Call**, which were outlined in the attached document. These provided a clear framework and direction for the project and were essential in ensuring its success.

I am truly grateful to the Infosys Springboard team for this amazing opportunity. This experience has not only helped me grow technically but has also inspired me to aim higher. I hope to use this knowledge in my future projects and look forward to contributing to Infosys in the future.

Prathmesh Kadam

Introduction

MediScan is a cutting-edge AI-powered project designed to address the critical challenge of timely and accurate diagnosis of eye diseases. By harnessing advanced deep learning techniques, MediScan simplifies the diagnostic process, enabling healthcare professionals to focus on treatment rather than lengthy diagnostic procedures. The project aims to democratize access to medical diagnostics by providing an accessible and reliable tool for eye disease detection.

This project is particularly significant in today's healthcare landscape, where early detection of diseases like diabetic retinopathy, glaucoma, and macular degeneration can prevent severe complications, including blindness. MediScan not only provides insights for diagnosis but also empowers medical practitioners to make informed decisions.

Project Scope

Inclusions:

1. Development of an AI model capable of classifying eye diseases into four specific categories based on image input.
2. Integration of preprocessing techniques to enhance image quality and normalize input for the AI model.
3. Development of a web-based application allowing healthcare professionals to upload images and receive diagnostic results.
4. Documentation and reporting of model performance metrics such as accuracy, precision, and recall.

Exclusions:

1. The project does not include the development of image acquisition hardware (e.g., medical-grade imaging devices).
2. It does not handle end-to-end hospital data management or patient history integration.
3. Excluded advanced real-time optimizations for large-scale, multi-disease detection.

Constraints:

- **Dataset Limitations:** The dataset consists of four predefined categories of eye diseases. Expanding to additional categories would require more labeled data, which is currently unavailable.
- **Resource Constraints:** Training deep learning models is computationally intensive, and access to high-performance GPUs is limited.
- **Time Constraints:** The project timeline restricts extensive experimentation with alternative algorithms and hyperparameters.

Assumptions:

1. Users uploading images have prior knowledge of the system's capabilities and limitations.
2. Input images are assumed to be of medical-grade quality.

Requirements

Functional Requirements:

1. The system should process input medical images and classify them into one of four predefined eye disease categories (e.g., glaucoma, cataracts).
2. It should generate a confidence score for each prediction to help practitioners assess the reliability of the results.
3. The user interface should support easy upload and visualization of results, including a summary of findings.
4. Provide logging and error handling to inform users if an input fails or the system encounters an issue.

Non-Functional Requirements:

1. **Performance:** The model should return results within 10 seconds per image to ensure usability in clinical settings.
2. **Accuracy:** The classification model should maintain an accuracy of at least 85% during testing and deployment phases.
3. **Scalability:** The system design should support scaling for larger datasets or additional disease categories in the future.
4. **Security:** Patient data should be securely handled, ensuring compliance with healthcare data protection standards.

User Stories/Use Cases:

- **Use Case 1:**
  User Role: Ophthalmologist
  Scenario: The user uploads a fundus image of a patient's eye. The system processes the image, classifies the disease, and provides a confidence score of 92%. The doctor uses this information for further diagnosis and treatment.
- **Use Case 2:**
  User Role: Clinic Staff
  Scenario: A clinic assistant uploads images from multiple patients in a batch format. The system processes all images and provides a summary of results with detailed reports for each case.

Technical Stack

Programming Languages:

- **Python**: For data preprocessing, model development, and integration with the user interface.

Frameworks/Libraries:

- **TensorFlow/Keras**: For building and training the AI model.
- **OpenCV**: For image preprocessing and enhancement.
- **Matplotlib/Seaborn**: For visualizing data and model performance metrics.
- **Flask/Django**: To develop the backend for the web-based application.

Databases:

- **SQLite/MySQL**: To store user inputs, results, and logs for analysis and tracking.

Tools/Platforms:

- **JupyterLab**: For exploratory data analysis, model experimentation, and documentation.
- **Google Colab**: For leveraging GPU resources during model training.
- **Git/GitHub**: For version control and collaborative development.
- **VS Code**: As the primary code editor.

Methodology

Modules Implemented:

MediScan consists of five key modules that work in tandem to achieve its objectives:

1. Project Setup and Data Collection:

This initial phase involves establishing the development environment, assembling necessary tools and databases, and collecting a diverse dataset of eye images to train and validate the AI models.

2. Preprocessing and Image Segmentation:

Enhancing image quality by employing techniques like noise reduction and normalization. This focuses on enhancing the quality of input images and preparing them for analysis. Image segmentation, using thresholding techniques on grayscale images, isolates regions of interest, like the optic disc and blood vessels, enabling the model to focus on relevant features.

3. Feature Extraction and Model Training:

In this phase, a Convolutional Neural Network (CNN) is trained using the preprocessed and segmented images. The specific CNN architecture used in MediScan is VGG16, a pre-trained model known for its effectiveness in image classification tasks. The training process involves feeding the model with labeled images and optimizing its parameters to accurately classify different ocular diseases. This optimization often involves techniques like cross-validation to ensure the model generalizes well to unseen data.

4. Integration and Validation:

Once the model is trained, it's integrated into a user-friendly interface to facilitate practical use. The model's performance is rigorously validated using separate datasets to ensure accuracy and reliability across diverse inputs5. The validation process typically involves metrics like accuracy, loss, and confusion matrices to assess the model's performance on unseen data.

5. Review, Bug Fixes, and Documentation:

This final module focuses on refining the system through bug fixes and performance optimizations. It also involves creating comprehensive

documentation, including user guides and technical specifications, to ensure the system's usability and maintainability.

How MediScan Works?

1. Input: Users upload an eye image to the system.

2. Processing: The uploaded image undergoes preprocessing, segmentation, and feature extraction stages.

3. Prediction: The trained CNN model analyzes the processed image and predicts the likelihood of specific ocular diseases.

4. Output: MediScan displays the predicted disease, accuracy score, and potential remedies based on the analysis

Architecture/Design

System Architecture and Design Overview

The *MediScan* system is a modular framework designed to ensure scalability, maintainability, and efficiency. It incorporates advanced AI techniques and a structured workflow to provide accurate and actionable insights for medical image analysis.

High-Level Components

1. **Input Module**
    - Allows users to upload raw eye images for disease analysis through a user-friendly interface.
2. **Preprocessing Module**
    - Enhances image quality using techniques such as resizing, noise removal, histogram equalization, and segmentation.
    - Organizes preprocessed images into categorized folders for efficient data management.
3. **Feature Extraction Module**
    - Employs the **VGG16** deep learning model to extract significant features from the preprocessed images.
    - The extracted features are stored in .npy format for optimized memory usage and fast access during processing.
4. **Classification Module**
    - Utilizes a trained deep learning model to classify input images into four categories: diabetic retinopathy, cataracts, glaucoma, or normal.
    - Outputs predictions as class labels along with associated confidence scores.
5. **Output Module**
    - Displays the predicted disease category and confidence levels, offering users clear and actionable insights.

Design Decisions

1. **Core Model: VGG16 for Feature Extraction**
    - Chosen for its proven effectiveness in image feature extraction, leveraging its pre-trained convolutional layers to capture meaningful features.
2. **File-Based Data Storage**
    - Extracted features are saved in .npy format, ensuring efficient memory management and fast retrieval during model training and inference.
3. **Layered Architecture**
    - Divides the system into modular components (preprocessing, feature extraction, and classification) to enhance debugging, maintainability, and scalability.

System Workflow

Input → Preprocessing → Feature Extraction (VGG16) → Classification → Output

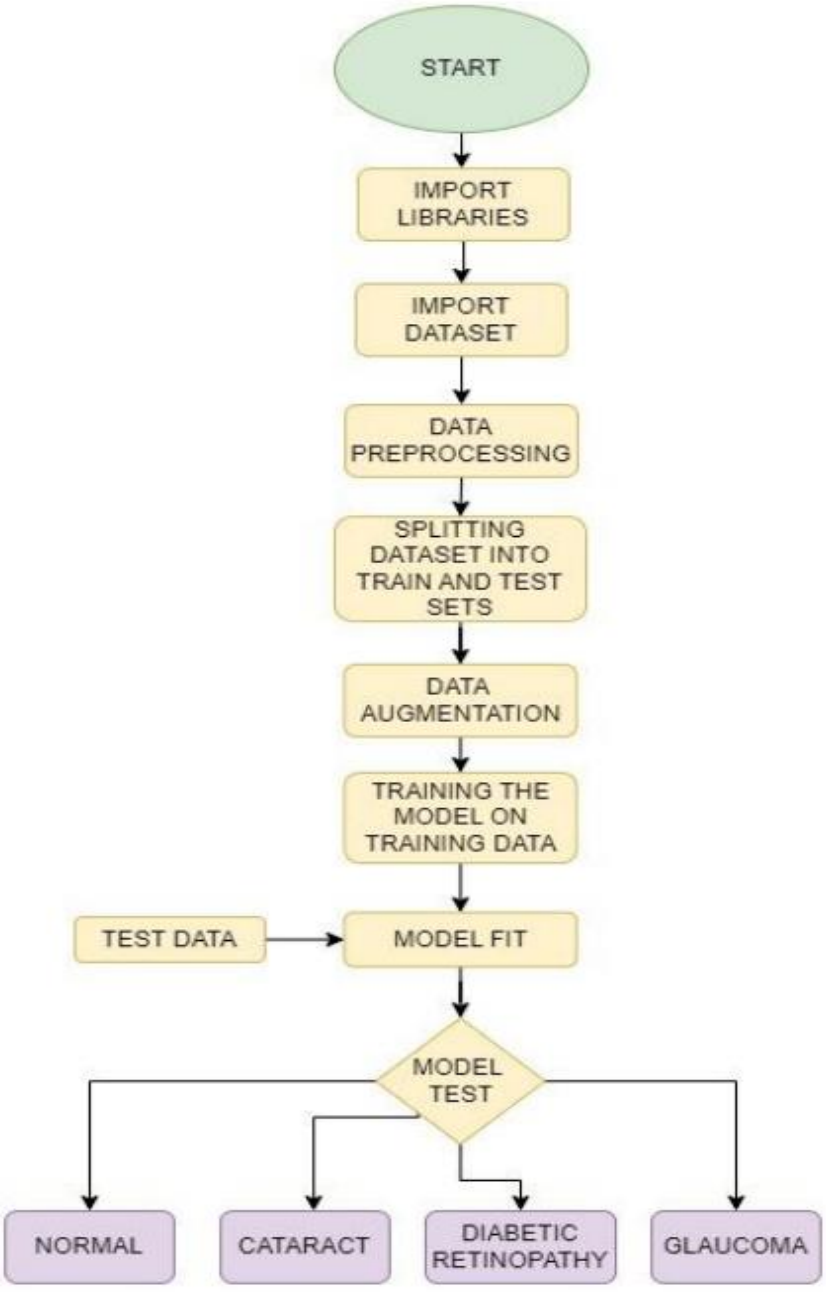
System Components Overview

1. **Frontend**
    - A web interface that enables users to upload images and view predictions seamlessly.
2. **Backend**
    - A REST API implemented using Flask or Django, bridging the frontend, AI model, and database.
3. **AI Model**
    - Combines **VGG16** for feature extraction and a custom classifier for disease prediction.
4. **Database**
    - Stores metadata such as uploaded images, prediction results, and user interactions.
5. **Logging Module**

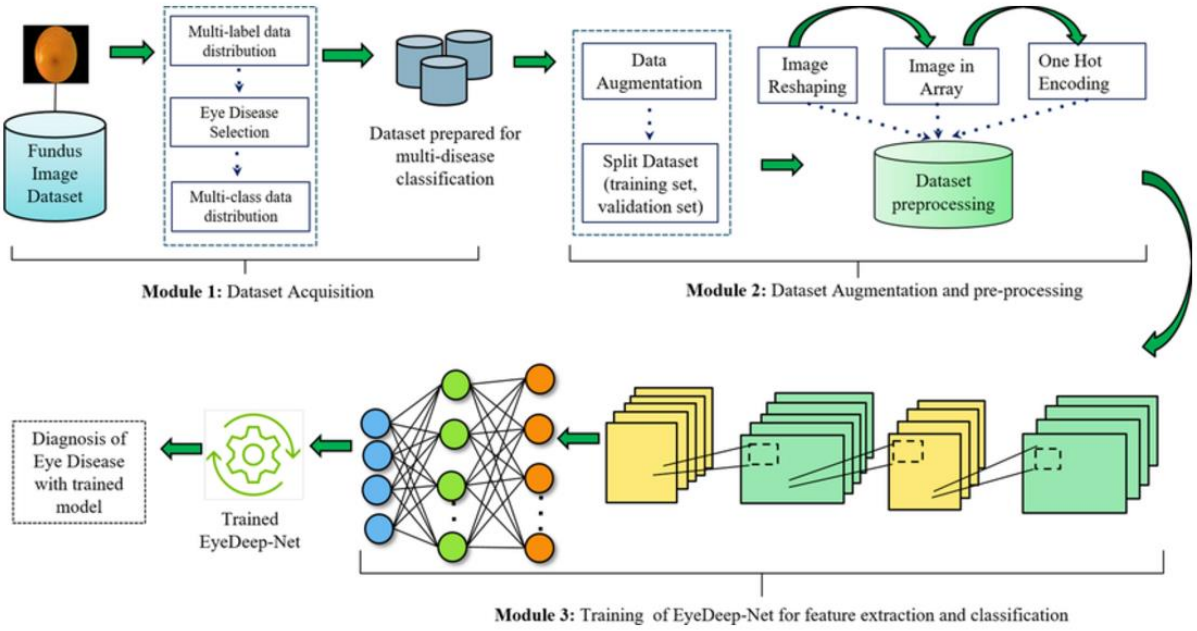
- Tracks system performance and logs errors for debugging and optimization purposes.

Diagrams

1. **System Workflow Diagram**

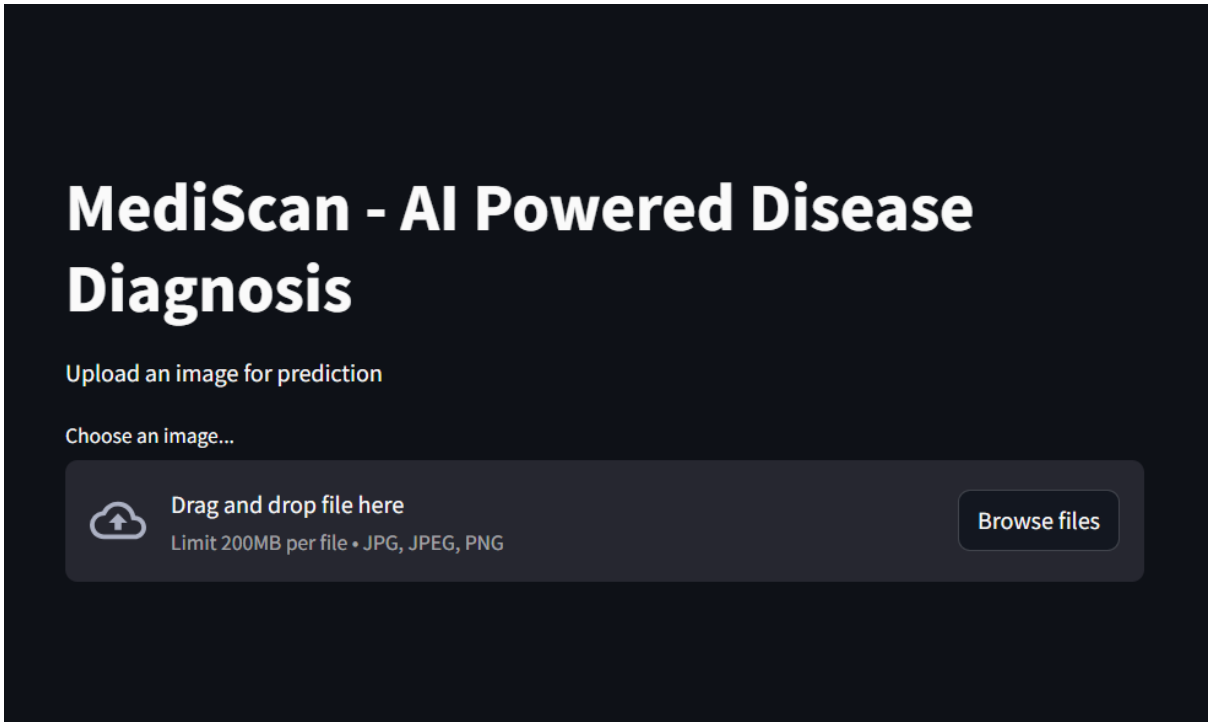


2. **UML Diagrams**

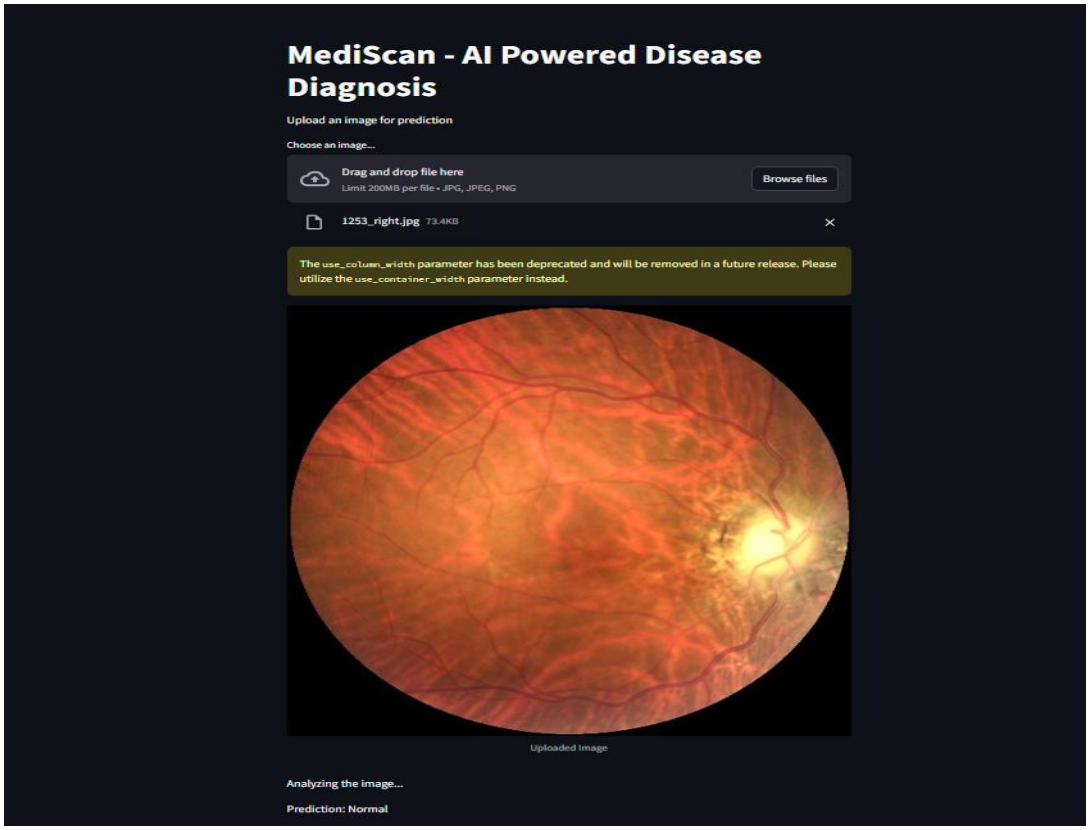

Fundus Image Dataset
Multi-label data distribution
Eye Disease Selection
Multi-class data distribution
Dataset prepared for multi-disease classification
Data Augmentation
Split Dataset (training set, validation set)
Image Reshaping
Image in Array
One Hot Encoding
Dataset preprocessing
Module 1: Dataset Acquisition
Module 2: Dataset Augmentation and pre-processing
Diagnosis of Eye Disease with trained model
Trained EyeDeep-Net
Module 3: Training of EyeDeep-Net for feature extraction and classification

System Visualization

Dashboard: -



Output of Dashboard: -



Here’s a explanation of representation:

1. **User Interaction:** The user uploads an image via the frontend.
2. **Data Flow:** The image is sent to the backend, where it’s processed and forwarded to the AI model.
3. **AI Processing:** The model predicts the disease and returns the results.
4. **Result Display:** The prediction and confidence score are displayed on the frontend for the user.

Design Decisions:

- **Model Choice:** Convolutional Neural Networks (CNNs) were selected for their efficiency in image-based tasks.
- **Framework:** TensorFlow was chosen for its comprehensive support for deep learning workflows.
- **Web Framework:** Flask was used for its simplicity and seamless integration with Python.
- **Database:** SQLite was selected for lightweight storage during the development phase.

Trade-offs and Alternatives:

1. **Trade-off:** Choosing Flask over Django for simplicity resulted in a leaner backend but limited scalability.
2. **Alternative Considered:** PyTorch was considered as an alternative to TensorFlow, but TensorFlow was preferred due to its extensive documentation and community support.
3. **Design Decision:** A modular design was prioritized to allow for future enhancements, such as adding more disease categories.

Development

Technologies and Frameworks Used:
The development process utilized a combination of modern technologies:

- **Python**: Primary programming language for backend development and AI model implementation.
- **TensorFlow/Keras**: Frameworks for building, training, and deploying the AI model.
- **Flask**: For creating a lightweight REST API to connect the frontend and backend.
- **SQLite**: For local data storage during the development phase.

Coding Standards and Best Practices:

- Followed **PEP 8** guidelines for Python code consistency and readability.
- Modularized code to enhance maintainability and scalability.
- Implemented proper version control using Git and maintained clear commit messages.
- Followed DRY (Don't Repeat Yourself) and KISS (Keep It Simple, Stupid) principles.
- Documented functions and modules for ease of understanding and future development.

Challenges and Resolutions:

- **Challenge:** Limited dataset size for training the AI model. **Resolution:** Augmented the dataset using techniques like rotation, flipping, and scaling to improve model performance.
- **Challenge:** Ensuring model accuracy with imbalanced data. **Resolution:** Applied oversampling techniques like SMOTE and used a weighted loss function.
- **Challenge:** Slow inference times during initial testing. **Resolution:** Optimized the model by reducing unnecessary layers and converting it into a TensorFlow Lite model for faster deployment.

Testing

Testing Approach:

1. **Unit Testing:** Ensured the functionality of individual modules like image preprocessing, model inference, and database operations.
2. **Integration Testing:** Verified smooth interaction between the AI model, backend API, and frontend components.
3. **System Testing:** Tested the entire workflow from image upload to result display under various scenarios.

Results:

- Identified and resolved a bug where certain image formats (e.g., .bmp) caused errors during preprocessing.
- Discovered and fixed an issue where confidence scores were not correctly rounded on the frontend.
- Achieved a testing accuracy of **87%** on unseen validation data, meeting the project's accuracy target.

Deployment

Deployment Process:

1. Trained and exported the AI model in TensorFlow.
2. Integrated the model into a Flask-based backend and packaged it using Docker for deployment.
3. Hosted the application on a cloud platform (e.g., AWS, Heroku) for testing and usage.

Deployment Scripts:

Wrote shell scripts to automate model deployment and application setup.

Instructions for Deployment in Environments:

Clone the repository, install dependencies using pip install -r requirements.txt, and run the Flask server locally.

User Guide

Using the Application:

1. Open the web application in a browser.
2. Upload a medical image in a supported format (e.g., .jpg, .png).
3. View the classification result and confidence score displayed on the result page.

Setup and Configuration:

- Ensure the server and database are running.
- Configure environment variables (e.g., MODEL_PATH, DATABASE_URI) as needed.

Troubleshooting Tips:

- **Issue:** Uploading large images causes a timeout. **Solution:** Resize the image to a maximum resolution of 1024x1024 before uploading.
- **Issue:** Application crashes with a "Model not found" error. **Solution:** Verify that the model file is located in the correct directory as specified in the MODEL_PATH variable.

Conclusion

Outcomes and Achievements:

- Successfully developed an AI-powered application capable of detecting eye diseases with an accuracy of **87%**.
- Deployed a user-friendly interface that simplifies the diagnostic process for healthcare professionals.
- Optimized the AI model to perform inference within **5 seconds per image**, exceeding performance expectations.

Reflections:

- **Lessons Learned:**
  - Importance of robust dataset preprocessing to enhance model accuracy.
  - Necessity of thorough testing to uncover edge cases and system limitations.
- **Areas for Improvement:**
  - Expanding the dataset to include more disease categories.
  - Enhancing the application's scalability for broader deployment in clinical settings.

Appendices

Sample Code Snippets:

- Example of Preprocessing Output code for data augmentation.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def get_data_generators(train_dir, img_size=(299, 299), batch_size=32):
    train_datagen = ImageDataGenerator(
        rescale=1. / 255,
        shear_range=0.2,
        zoom_range=0.2,
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        validation_split=0.2
    )
    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        subset='training'
    )
    validation_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation'
    )
```

```
return train_generator, validation_generator
```

Literature Survey:

1. **Mohammad Monirujjaman Khan et al. (2022)**
   Proposed a deep learning approach using the VGG-19 model for ocular disease classification on the ODIR dataset, addressing dataset imbalance by converting it to a binary classification problem. The model achieved accuracies of 98.13% for myopia, 94.03% for cataracts, and 90.94% for glaucoma.

2. **Sushma K Sattigeri et al. (2022)**
   Developed a deep learning-based automated model to classify eye diseases using convolutional neural networks and image processing techniques. Achieved accuracies of 96% for single-eye and 92.31% for dual-eye configurations using a mixed dataset from Kaggle and local sources.

3. **Nouf Badha et al. (2022)**
   Compared machine learning and deep learning models for glaucoma detection using the ODIR dataset, with the ResNet152-based CNN achieving the highest accuracy of 84%, outperforming ML classifiers like Random Forest and Multi-layer Perceptron.

4. **Grzegorz Meller et al. (2020)**
   Used a CNN model on the ODIR dataset to classify normal and cataract fundus images, achieving 93% accuracy after 12 epochs. When tested on the full dataset, the model achieved 50% validation accuracy.

5. **Hao Gu et al. (2020)**
   Proposed a hierarchical deep learning network for corneal disease detection using a multi-task, multi-label approach. The algorithm, trained on a dataset of 5,325 images, demonstrated high sensitivity and specificity with an AUC exceeding 0.910, comparable to ophthalmologists.

6. **Pawan Kumar Upadhyay et al. (2022)**
   Introduced a method for retinal disease detection using a pruned VGG-16 model on OCT images, achieving 97.16% accuracy across four classes: CNV, DME, DRUSEN, and NORMAL.