A MINI PROJECT

On

EYE DISEASE CLASSIFICATION AND PREDICTION USING CNN

Submitted

In partial fulfillment for the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science and Engineering (Data Science)

By

B. SHIVA 21641A67E8

Under the Guidance of

Mr. SYED HASANODDIN

Assistant Professor, Dept. of CSE (Data Science).

Department of Computer Science & Engineering (Data science)

Vaagdevi College of Engineering

(UGC Autonomous, Accredited by NAAC with "A")
Bollikunta, Khila Warangal (Mandal), Warangal Urban – 506005(T.S)

(2021-2025)

VAAGDEVI COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(DATA SCIENCE)

(UGC Autonomous, Accredited by NBA, Accredited by NAAC with "A")

Bollikunta, Khila Warangal (Mandal), Warangal Urban –506005(T.S)

CERTIFICATE

This is to certify that the mini project entitled "***EYE DISEASE CLASSIFICATION AND PREDICTION USING CNN"*** is submitted by **B. SHIVA (21645A67E8** in partial fulfillment of the requirements for the award of the Degree in Bachelor of Technology in Computer Science and Engineering(Data Science) during the academic year 2024-2025.

Project Guide:
Mr. Syed Hasanoddin

Head of the Department:
Dr. Ayesha Banu

External Examiner

DECLARATION

We declare that the work reported in the project entitled "EYE DISEASE CLASSIFICATION AND PREDICTION USING CNN" is a record of work done by us in the partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science and Engineering(Data Science) ,VAAGDEVI COLLEGE OF ENGINEERING (Autonomous), Affiliated to JNTUH, Accredited By NBA, under the guidance of **Dr. Ayesha Banu** , Associate Professor, HOD of CSE (DS). We hereby declare that this project work bears no resemblance to any other project submitted at Vaagdevi College of Engineering or any other university/college for the award of the degree.

Bilavath Shiva (21641A67E8)

ACKNOWLEDGEMENT

The development of the project though it was an arduous task, it has been made by the help of many people. We are pleased to express our thanks to the people whose suggestions, comments, criticisms greatly encouraged us in betterment of the project.

We would like to express our sincere gratitude and indebtedness to my project Guide **Mr. SYED HASANODDIN**, Assistant Professor, CSE (DS) for her valuable suggestions and interest throughout the completion of this project.

We are also thankful to Head of the Department **Dr. Ayesha Banu**, Associate Professor, CSE (DS) for providing excellent support in completing the project successfully.

We would like to express our sincere thanks and profound gratitude to **Dr. K. Prakash**, Principal of Vaagdevi College of Engineering, for his support, guidance and encouragement in the course of our project.

We are also thankful to Project Coordinators, for their valuable suggestions, encouragement and motivations for completing this project successfully.

We are thankful to all other faculty members for their encouragement.

Finally, we would like to take this opportunity to thank our family for their support through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

Bilavath Shiva (21641A67E8)

ABSTRACT

Eye diseases such as glaucoma, cataracts, and diabetic retinopathy are leading causes of vision impairment and blindness worldwide. Early diagnosis and timely treatment are critical in preventing vision loss. Traditional diagnostic methods are often time-consuming, resource-intensive, and highly dependent on the availability of specialized medical professionals. This project aims to automate the diagnosis of common eye diseases using Convolutional Neural Networks (CNNs), leveraging the power of deep learning in image analysis.

CNNs are well-suited for medical image classification due to their ability to automatically extract and learn relevant features from raw pixel data. In this study, a CNN model is trained on a dataset of retinal images to classify and predict the presence of various eye conditions. Additionally, machine learning techniques, such as Support Vector Machines (SVM), are considered for enhanced classification performance. The system's effectiveness is evaluated using metrics like precision, recall, F1-score, accuracy, and support, providing a robust measure of diagnostic accuracy.

This project also explores the use of semi-supervised learning, combining labeled and unlabeled data, to improve the model's performance and reduce the dependency on large, fully labeled datasets. The proposed model has potential applications in telemedicine, offering accessible, accurate, and efficient eye disease screening, especially in underserved regions. Future improvements could include expanding the dataset and refining the model architecture to increase diagnostic accuracy and reliability, ultimately contributing to better global eye health outcomes.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Eye diseases, including glaucoma, cataracts, and diabetic retinopathy, are significant contributors to vision impairment and blindness, which can severely impact quality of life. Early detection and treatment are essential to prevent the progression of these conditions, but traditional diagnostic processes are often resource-intensive, requiring specialized equipment and trained ophthalmologists. This makes access to timely and effective eye care challenging, particularly in remote and underserved areas.

With the rapid advancement of artificial intelligence and deep learning, there is growing interest in leveraging these technologies to support medical diagnostics. Convolutional Neural Networks (CNNs), a type of deep learning architecture, have proven effective in analyzing and classifying medical images, particularly in the field of ophthalmology. CNNs are capable of automatically learning relevant features from images, making them suitable for identifying patterns associated with different eye diseases.

This project aims to create a CNN-based system for the classification and prediction of eye diseases, focusing on automating the diagnosis process to improve accessibility and efficiency. By training on a dataset of retinal images, the model learns to recognize disease-specific features, potentially supporting healthcare providers with more accurate and faster diagnostic tools. The project also evaluates additional machine learning algorithms, such as Support Vector Machines (SVM), to complement CNN classification results and enhance performance.

To address the challenge of limited labeled data in medical image analysis, the project also explores semi-supervised learning, which combines a small set of labeled data with a larger set of unlabeled data. This approach could help improve the model's generalization and robustness, making it better suited for real-world applications.

This automated diagnostic system has the potential to be integrated into telemedicine platforms, offering remote and accessible eye care. The project ultimately seeks to advance the use of deep learning in healthcare, contributing to more proactive and preventive care in ophthalmology, and supporting global efforts to reduce preventable blindness.

1.1 EXSISTING SYSTEM

- **Manual eye examination:** Doctors examine retinal images using specialized equipment like fundus cameras, which capture images of the retina. Ophthalmologists analyze these images visually to identify abnormalities or early signs of diseases like cataracts, glaucoma, and diabetic retinopathy.

- **Fundus Photography and Fluorescein Angiography**: These methods help in diagnosing eye diseases by producing detailed images of the blood vessels in the retina and determining blood flow patterns. These techniques are manually interpreted by professionals.

- **Lack of Automation**: The current systems heavily rely on the expertise of healthcare professionals for image analysis, which can be time-consuming and prone to human error.

- **Limited Reach in Remote Areas**: Access to eye specialists and diagnostic facilities is limited in rural or underdeveloped regions, leading to delayed or missed diagnoses.

LIMITATIONS

- **Data Quality and Quantity:** Requires large amounts of high-quality labeled data; poor data can lead to inaccurate predictions.

- **SVM:** Support Vector Machine Algorithm can have a more Time Comlexity, instead of SVM use the Random Forest Algorithm (RFA).

- **Dependence on Technology:** Over-reliance on automated systems may weaken critical thinking and diagnostic skills among healthcare professionals.

- **Regulatory and Ethical Issues:** Raises concerns about data privacy, informed consent, and accountability for diagnostic errors.

- **Cost of Implementation:** Initial setup costs for software, hardware, and training can be high, potentially limiting accessibility.

1.2 PROPOSED SYSTEM

- **Automated Image Classification:** The system processes retinal images and classifies them into categories such as normal, glaucoma, cataract, and diabetic retinopathy, reducing reliance on manual examination by specialists.

- **Use of CNN**: CNNs are specifically designed to handle image data, making them ideal for extracting patterns and features from medical images. The model identifies disease-related features in retinal images automatically.

- **Data Preprocessing**: The system preprocesses the input images by resizing, normalizing pixel values, and applying techniques like data augmentation to improve model generalization and accuracy.

- **Continuous Learning**: The model can continuously improve with more training data, making it adaptable to new patterns and better equipped for evolving medical needs.

- **Faster Diagnosis**: Unlike manual analysis, the automated system can quickly process and classify large volumes of retinal images, making diagnosis faster and more efficient.

- **Real-Time Prediction**: Once deployed, the system can provide real-time disease prediction and classification, assisting in timely medical intervention.

ADVANTAGES

- **Early Detection:** Facilitates timely diagnosis, reducing the risk of vision loss through prompt treatment.

- **Increased Accuracy**: Enhances diagnostic precision by identifying subtle patterns that may be missed by human observers.

- **Efficiency:** Speeds up the diagnosis process, allowing healthcare professionals to focus more on patient care.

- **Accessibility**: Improves screening in underserved areas through remote diagnosis and telemedicine, reaching more patients.

- **Cost-Effectiveness:** Lowers healthcare costs by reducing the need for extensive manual assessments and processing large volumes of images quickly.

1.3 SOFTWARE REQUIREMENTS

- **Programming Language**
  Python is the primary programming language used for this project due to its extensive library support for machine learning and deep learning. It provides simplicity and flexibility in implementing CNN models. Python's robust community and documentation make it ideal for troubleshooting and innovation. The project's code will include image preprocessing, model training, and evaluation tasks. Python also supports integration with other tools and frameworks for seamless development.

- **Integrated Development Environment (IDE)**
  An IDE like Jupyter Notebook, PyCharm, or VS Code is essential for writing, debugging, and testing the code. Jupyter Notebook is particularly suitable for step-by-step development and visualization of results. PyCharm provides advanced debugging features and support for large-scale projects. VS Code is lightweight and supports Python extensions for better development experience. These tools ensure efficient development and management of the project**.**

- **Libraries and Frameworks**
  Libraries such as NumPy and Pandas are crucial for numerical and data manipulation tasks. Matplotlib and Seaborn help visualize the dataset and model performance. OpenCV handles image preprocessing like resizing and augmentation. Deep learning frameworks like PyTorch or Keras with TensorFlow simplify CNN implementation and training. Scikit-Learn is used for calculating metrics like precision, recall, and F1-score.

- **Operating System**
  The project can be developed on Windows, macOS, or Linux, based on developer preference. Windows is widely used for compatibility with various software, while Linux offers better performance for machine learning tasks. macOS provides a stable and user-friendly environment for AI development. The operating system should support necessary software installations and GPU integration. Ensuring compatibility with CUDA-enabled GPUs is vital for faster training.

- **Image Preprocessing Tools**
  Tools like OpenCV and image augmentation libraries are essential for preparing the dataset. Preprocessing involves resizing images, normalizing pixel values, and applying augmentations like rotations or flips. This ensures the model is trained on a diverse and consistent dataset. Image preprocessing improves the accuracy and generalization of the CNN model. These tools are integrated into Python for seamless functionality.

1.4 HARDWARE REQUIREMENTS

- **Processor (CPU)**
  A high-performance multi-core processor such as Intel Core i7/i9 or AMD Ryzen 7/9 is recommended. These CPUs ensure smooth execution of preprocessing tasks, model training, and real-time predictions. For smaller datasets, mid-range processors like Intel Core i5 or AMD Ryzen 5 may suffice.

- **RAM (Memory)**
  At least 16GB of RAM is recommended for smooth performance during data preprocessing and model training. For larger datasets, 32GB or more is ideal to handle memory-intensive operations and avoid bottlenecks. Sufficient RAM ensures multitasking without lags or crashes.

- **Storage**
  A combination of SSD and HDD is recommended for optimal performance. SSD (at least 512GB) is essential for faster read/write operations during data loading and processing. HDD (1TB or more) can be used for storing large datasets, models, and backups.

- **Monitor**
  A high-resolution monitor (Full HD or above) is required for clear visualization of retinal images. Large monitors or dual-monitor setups improve productivity during development and debugging.

- **Power Supply Unit (PSU)**
  A reliable PSU with sufficient wattage to support the GPU and other components is crucial. For systems with high-end GPUs, a 750W or higher PSU is recommended.

- **Cooling System**
  A proper cooling solution, such as liquid cooling or high-performance air coolers, is necessary. This prevents overheating during prolonged GPU-intensive tasks like model training.

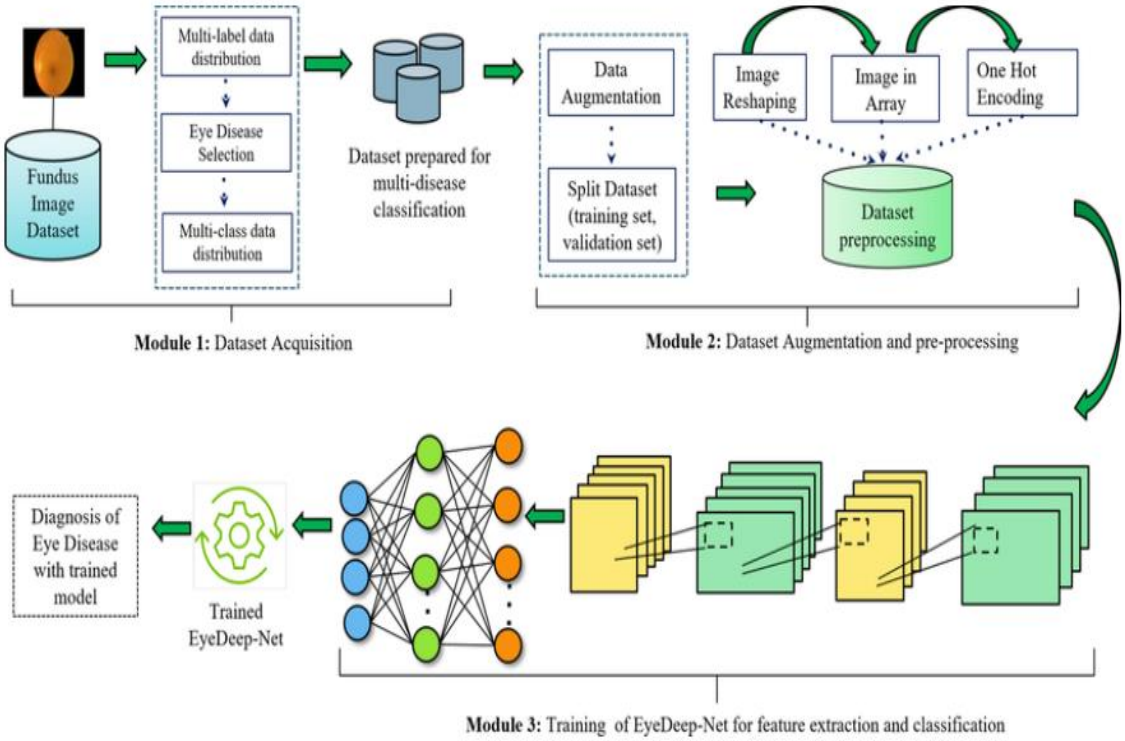
- **Peripheral Devices**
  Keyboard and Mouse are comfortable and responsive peripherals for efficient coding. External Storage are portable hard drives or SSDs (1TB or more) for backing up datasets and models.

- **Network**
  A stable internet connection is essential for downloading datasets, libraries, and pre-trained models. For cloud-based training or deployment, high-speed broadband is recommended.

CHAPTER 2

DESIGN OF THE PROJECT

2.1 FLOW CHART



2.2 ALGORITHM

Step 1: Import Required Libraries

Import libraries for image processing, data handling, machine learning, and evaluation:

- os, cv2, numpy, pandas
- sklearn modules for preprocessing, model building, and evaluation
- matplotlib for visualization

Step 2: Define Constants

- Set the path to the dataset directory (dataset_dir).
- Define the target image size (IMAGE_SIZE) for resizing images.

Step 3: Load and Preprocess Images

load_images_from_directory(directory, img_size=IMAGE_SIZE)

- Initialize empty lists for images and labels.
- Iterate through the subdirectories (representing class labels).
- For each image:
  - Load using cv2.imread.
  - Resize to IMAGE_SIZE and normalize pixel values to range [0, 1].
  - Append the processed image and its label to the respective lists.
- Return the images (as a NumPy array) and labels (as a NumPy array).

Step 4: Prepare Dataset

- Load images and labels using the defined function.
- Reshape image data into a 2D array (flattened_images) for model input.

Step 5: Encode Labels

- Use LabelEncoder to convert string labels to numeric values (train_labels_encoded).

Step 6: Split Data

- Split data into training and validation sets (X_train, X_val, y_train, y_val) using train_test_split (e.g., 80% training, 20% validation).

Step 7: Scale Features

- Apply StandardScaler to normalize features to a standard scale.

Step 8: Dimensionality Reduction (Optional)

- Use PCA to reduce feature dimensions to a manageable size (e.g., 50 components).
- Transform training and validation sets into the reduced PCA space.

Step 9: Train a Classifier

- Choose a model for classification:
- Random Forest Classifier:
  - Train using RandomForestClassifier with desired hyperparameters.
- Alternatively, Support Vector Machine (SVM) can also be used.
- Fit the model to the training data.

Step 10: Validate the Model

- Make predictions on the validation set (y_pred).
- Evaluate performance using metrics:
  - Accuracy: accuracy_score.
  - Detailed Report: classification_report.

Step 11: Predict New Image

- Preprocess the new image:
  - Load, resize, normalize, and flatten.
  - Apply the scaler and PCA transformations.
- Predict the class using the trained model.
- Decode the predicted label using LabelEncoder.inverse_transform.

Step 12: Output Results

- Display the validation accuracy and classification report.
- Output the predicted class for the new image.

2.3 UML DIAGRAMS

A UML (Unified Modeling Language) diagram is a standardized visual representation used to model and design the structure or behavior of a system. It helps developers and stakeholders understand, analyze, and document system requirements and functionalities. Common types include use case diagrams, class diagrams and sequence diagrams,

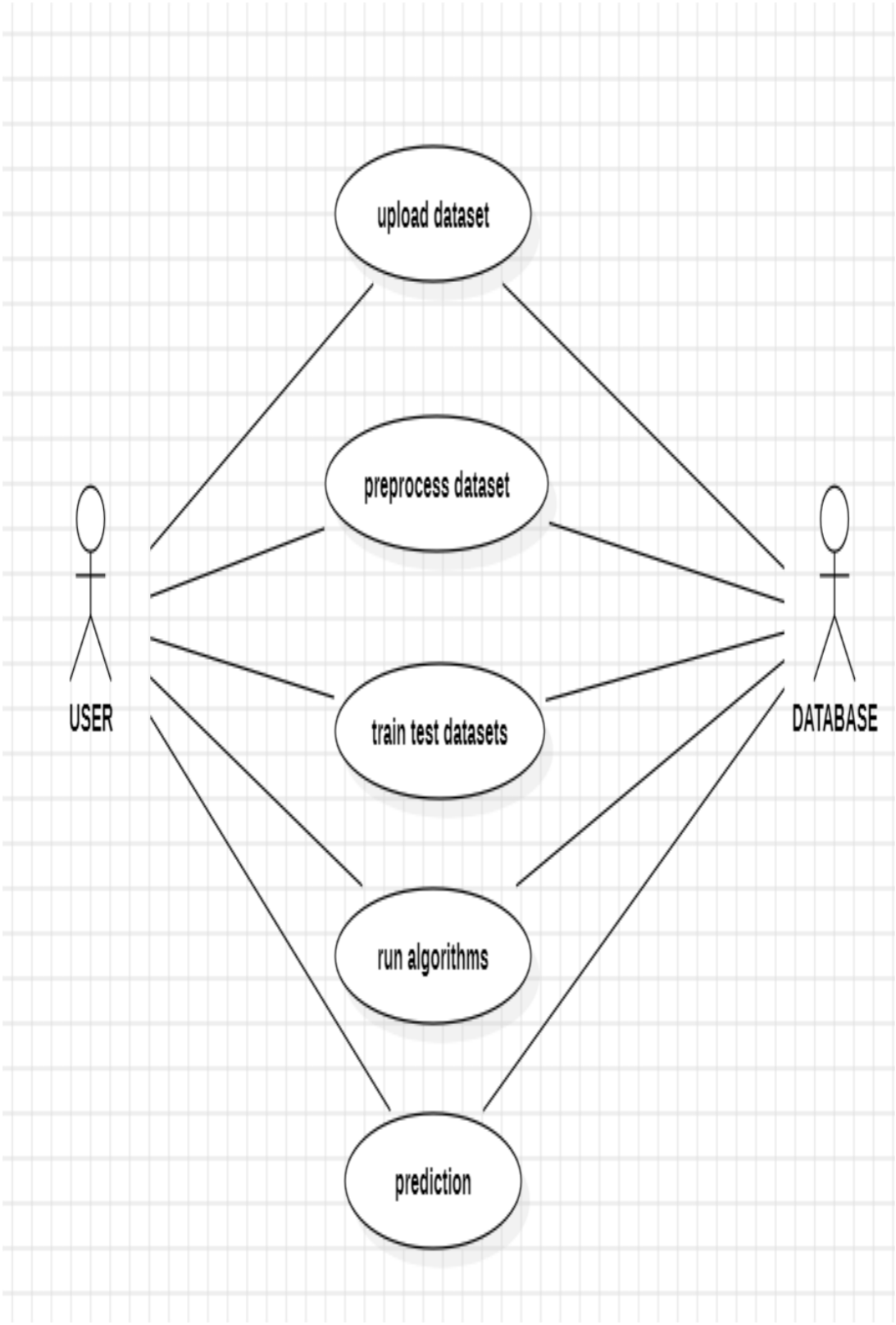
USECASE DIAGRAM

Actors

- User: A person (like a doctor, researcher, or patient) interacting with the system.
- Database: The storage system holding the dataset and results.

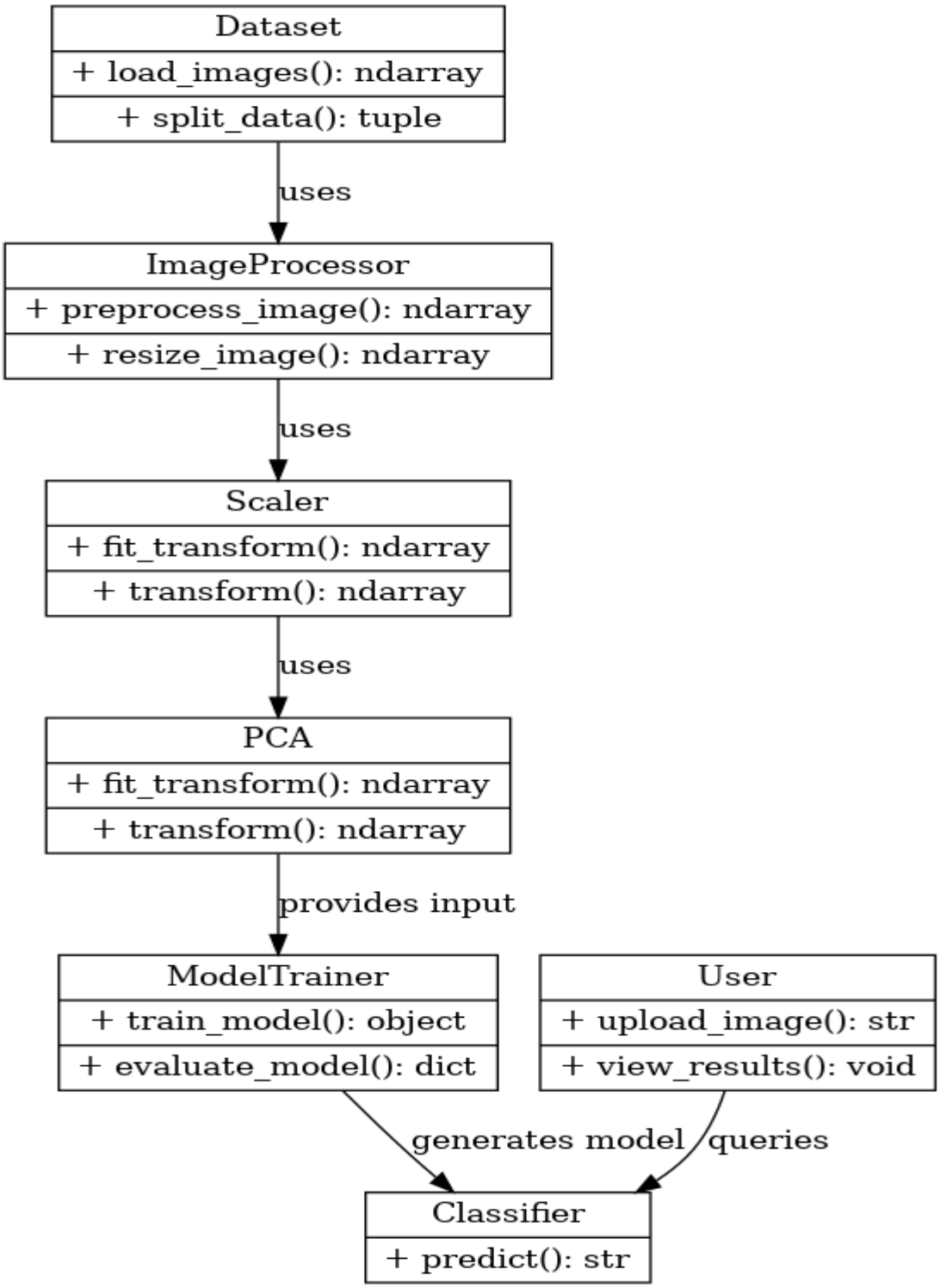
Use Cases

1. **Upload Dataset:**
   - Actor: User
   - The uploaded images are preprocessed to enhance quality, resize dimensions, normalize pixel values, and prepare them for model training.
2. **Train-Test Datasets:**
   - Actor: User
   - The preprocessed dataset is split into training and testing subsets.
   - Training data is used to build the CNN model, while testing data is reserved for evaluation.
3. **Run Algorithms:**
   - Actor: System (automatically triggered by the user)
   - CNN (Convolutional Neural Network) algorithms are applied to train the model.
   - The system learns to classify images into their respective disease categories based on patterns in the data.
4. **Prediction:**
   - Actor: User
   - The user inputs a new eye image into the system.
   - The trained CNN model predicts whether the image corresponds to "normal" or a specific eye disease (e.g., glaucoma, diabetic retinopathy, cataract).
   - The prediction result is displayed to the user and stored in the database.


upload dataset
preprocess dataset
USER
train test datasets
DATABASE
run algorithms
prediction

CLASS DIAGRAM

1. **User**
   - **Attributes:**
     - userID: Unique identifier for the user.
     - username: Name of the user.
     - role: Role of the user (e.g., doctor, researcher).
   - **Methods:**
     - uploadDataset(): Allows the user to upload an eye disease dataset.
     - inputImage(): Allows the user to provide a new eye image for prediction.
     - viewPrediction(): Displays the prediction results to the user.
2. **Dataset**
   - **Attributes:**
     - datasetID: Unique identifier for the dataset.
     - imageData: Contains image files (e.g., JPG, PNG).
     - labels: Associated disease labels (normal, glaucoma, diabetic retinopathy, cataract).
   - **Methods:**
     - preprocessData(): Normalizes and preprocesses images for the CNN model.
     - splitData(): Splits the dataset into training and testing sets.
3. **CNNModel**
   - **Attributes:**
     - modelID: Unique identifier for the CNN model.
     - architecture: Details of the CNN layers (e.g., convolutional layers, pooling layers).
     - trainingStatus: Tracks if the model is trained or not.
     - trainModel(): Trains the CNN model on the training dataset.
     - testModel(): Evaluates the CNN model on the test dataset.
     - predictDisease(): Predicts the category of an input image.


Dataset
+ load_images(): ndarray
+ split_data(): tuple
uses
ImageProcessor
+ preprocess_image(): ndarray
+ resize_image(): ndarray
uses
Scaler
+ fit_transform(): ndarray
+ transform(): ndarray
uses
PCA
+ fit_transform(): ndarray
+ transform(): ndarray
provides input
ModelTrainer
+ train_model(): object
+ evaluate_model(): dict
User
+ upload_image(): str
+ view_results(): void
generates model
queries
Classifier
+ predict(): str

SEQUENCE DIAGRAM

Actors and Objects in the Sequence Diagram

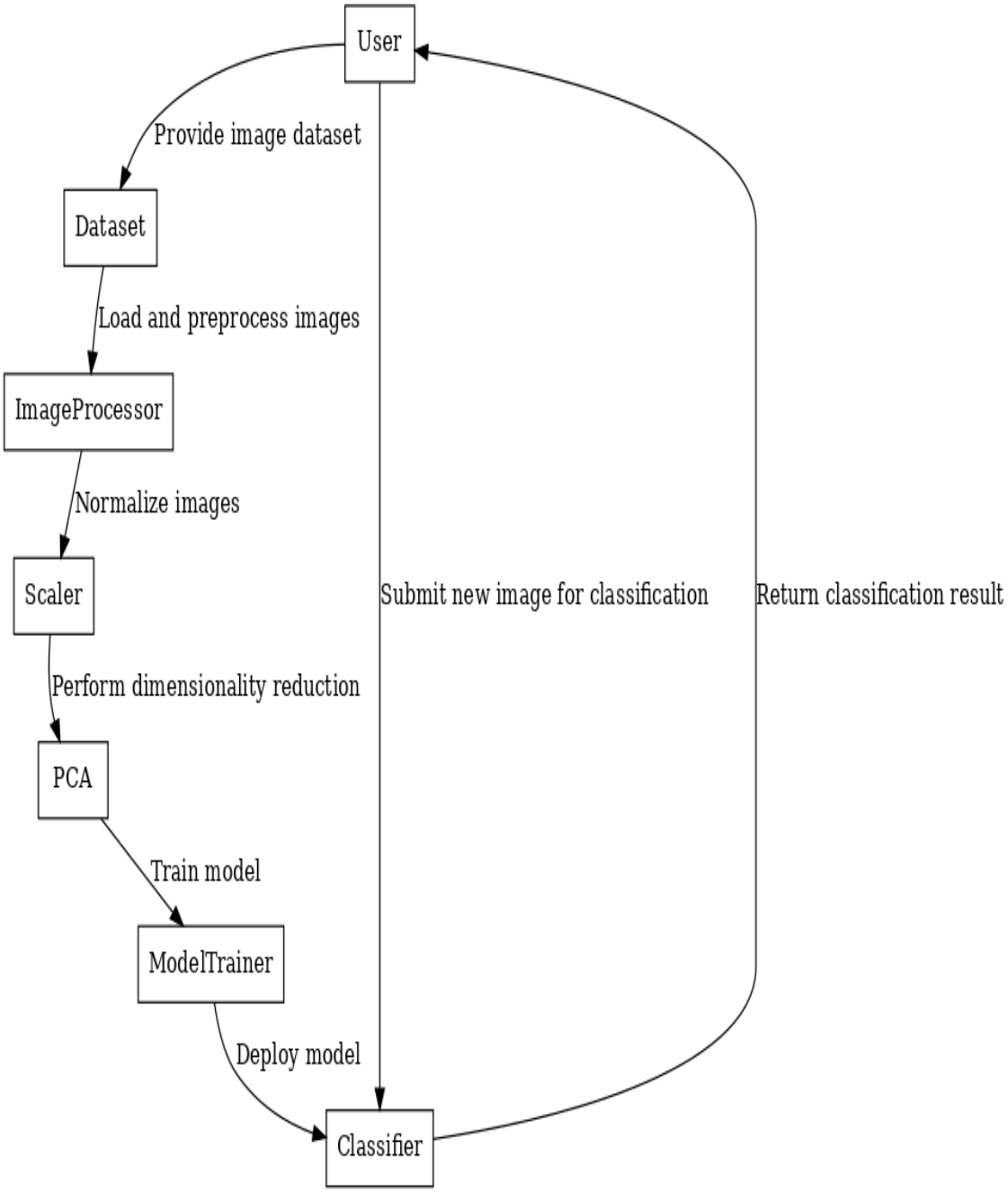
1. User: Initiates the interaction with the system by providing the dataset or a new image for classification.
2. Dataset: Contains the eye disease images used for training and testing.
3. ImageProcessor: Handles loading, preprocessing, and normalizing the images.
4. Scaler: Adjusts image scales (e.g., resizing or standardization for the model).
5. PCA (Principal Component Analysis): (Optional) Performs dimensionality reduction to optimize feature extraction.
6. Classifier: Deploys the trained model to classify new input images and return results to the user.

Sequence of Interactions

1. Provide Image Dataset:
    - The User uploads an image dataset containing eye disease images.
    - The Dataset object stores and organizes the uploaded data.
2. Load and Preprocess Images:
    - The ImageProcessor loads the dataset images and applies preprocessing steps, such as resizing, cleaning, and enhancing image quality.
3. Normalize Images:
    - The Scaler normalizes the images (e.g., scaling pixel values between 0 and 1) to ensure consistent input for the CNN model.
4. Perform Dimensionality Reduction:
    - PCA (if included) reduces the dataset's dimensionality by extracting the most important features, making the training process more efficient.
5. Train Model:

- The ModelTrainer uses the preprocessed and reduced dataset to train the CNN model. It iterates over the data to optimize model weights and parameters.

6. Deploy Model:
    - Once trained, the Classifier deploys the CNN model, making it ready for real-time classification tasks.



CHAPTER 3

IMPLEMENTATION

SOURCE CODE :

```
import os
import cv2
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

dataset_dir = "C:/Users/SHIVA/Downloads/eye disease dataset/dataset"
IMAGE_SIZE = (128, 128)  # Resize images to 128x128 pixels

def load_images_from_directory(directory, img_size=IMAGE_SIZE):
    images = []
    labels = []
    class_names = os.listdir(directory)

    for class_name in class_names:
        class_dir = os.path.join(directory, class_name)
        if os.path.isdir(class_dir):
            for img_name in os.listdir(class_dir):
                img_path = os.path.join(class_dir, img_name)
                img = cv2.imread(img_path)
                if img is not None:
                    img = cv2.resize(img, img_size)
                    img = img / 255.0  # Normalize pixel values
```

```
                images.append(img)
                labels.append(class_name)

    return np.array(images), np.array(labels)

# Load training images and labels
#train_images, train_labels = load_images_from_directory(dataset_dir)
# Load a smaller subset for quick testing
train_images, train_labels = load_images_from_directory(dataset_dir)[:1000]

# Check shapes
print("Train Images Shape:", train_images.shape)
print("Train Labels Shape:", train_labels.shape)

n_samples = len(train_images)
flattened_images = train_images.reshape(n_samples, -1)

# Label encoding
le = LabelEncoder()
train_labels_encoded = le.fit_transform(train_labels)

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(flattened_images,
train_labels_encoded, test_size=0.2, random_state=42)

print("Training Data Shape:", X_train.shape)
print("Validation Data Shape:", X_val.shape)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)

# Optional: Use PCA for dimensionality reduction
pca = PCA(n_components=50)
```

```
X_train_pca = pca.fit_transform(X_train)
X_val_pca = pca.transform(X_val)
print("PCA-Reduced Training Data Shape:", X_train_pca.shape)

# Train a Support Vector Machine (SVM) classifier (or) Random Forest Classifier
#svm_classifier = SVC(kernel='linear', probability=True)
#svm_classifier.fit(X_train_pca, y_train)
# Train a Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_pca, y_train)

# Make predictions on the validation set
y_pred = rf_classifier.predict(X_val_pca)
# Print accuracy and classification report
accuracy = accuracy_score(y_val, y_pred)
print(f"Validation Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_val, y_pred, target_names=le.classes_))

# Path to new image for prediction (replace with your own path)
img_path = "C:/Users/SHIVA/Downloads/eye disease dataset/online_img.jpg"

# Preprocess the image
img = cv2.imread(img_path)
img = cv2.resize(img, IMAGE_SIZE)
img = img / 255.0  # Normalize
img_flattened = img.reshape(1, -1)
img_scaled = scaler.transform(img_flattened)
img_pca = pca.transform(img_scaled)

# Predict the class of the new image
predicted_label_index = rf_classifier.predict(img_pca)
predicted_label = le.inverse_transform(predicted_label_index)
print("Predicted class:", predicted_label[0])
```

CHAPTER 4

TESTING

4.1 BLACK BOX TESTING

Black box testing focuses on the functionality of the model without considering its internal workings (i.e., the model's code or algorithm). You evaluate the model purely based on its input-output behavior. This is especially relevant for machine learning projects where you are interested in testing the performance and correctness of the model from a user perspective.

1. Functional Testing

- Ensure that the model produces valid predictions when given valid inputs (e.g., images of eye diseases). The output should be a valid classification (e.g., "glaucoma," "diabetic retinopathy," etc.).
- **Example**: Provide the model with an image of a normal eye, and verify that the model outputs the correct prediction (e.g., "normal").

2. Boundary Value Testing

- Test the model with extreme or edge-case inputs to ensure it handles them correctly.
- **Example**: Give the model images with very low resolution or very high resolution and check how the model handles them.

3. Input Validation Testing

- Test how well the model handles invalid or unexpected inputs.
- **Example**: Input an image that doesn't belong to any eye disease category (e.g., a landscape) and check if the model rejects it or returns an error.

4. Usability Testing

- Assess how easy it is for a user to interact with the model. For example, the model might be integrated into a web interface, so you can test whether it is easy for users to upload images and get results.
- **Example**: Ensure that the model can handle different image formats (JPG, PNG) and provides an understandable output (e.g., a clear label or confidence score).

5. Performance Testing

- Test how the model performs in terms of speed, memory usage, and other system resources when making predictions.
- **Example**: Test the model's prediction time on a large batch of images and see if it performs efficiently under load.

4.2 WHITE BOX TESTING

White box testing involves testing the internal logic and structure of the model. This type of testing requires knowledge of the code and algorithm to ensure everything is functioning as expected at the code level.

1. Code Review

- Review the model's code to identify potential issues like unhandled exceptions, poor error handling, or inefficient algorithms.
- **Example**: Ensure the data preprocessing steps (like normalization, resizing) are implemented correctly. Look for any logical errors in the code.

2. Path Testing

- Test all possible paths through the code to ensure that every conditional statement (e.g., if, else) and loop (e.g., for, while) is executed at least once.
- **Example**: If your code handles different disease classes (e.g., normal, glaucoma, cataract), check that the logic covers all the classes and conditions that could arise.

3. Unit Testing

- Write unit tests for individual functions or components in the system to ensure that each part of the code works as expected.
- **Example**: Test a function that pre-processes images to ensure it works correctly when handling different image sizes, formats, and noise.

4. Regression Testing

- Ensure that new changes to the model or code don't break any existing functionality. You can compare the model's output before and after making updates.
- **Example**: After adding new features (like a new pre-processing step), test that the model's prediction accuracy does not decrease unexpectedly.

5. Security Testing

- Verify that the model and its environment are secure and that no unauthorized access or data leaks can occur.
- **Example**: Ensure that the input images and the results are kept confidential, especially if the model is deployed in a healthcare setting.

6. Algorithm Testing

- Test the underlying machine learning algorithm (e.g., decision tree, CNN) to ensure it is properly tuned, optimized, and functioning as expected.
- **Example**: Evaluate whether the correct loss function is being used and if the model converges properly during training.

4.3 GRAY BOX TESTING

1. Dataset Upload Testing

The dataset upload process is critical to ensure the system receives properly formatted and structured data for training and predictions. This module is tested to verify that the system can handle various scenarios such as uploading valid datasets in supported file formats like .jpg and .png. The system should reject incomplete datasets, such as those missing labels or essential images, with appropriate error messages guiding the user. Unsupported file types like .txt or .gif should be rejected as well.

2. Preprocessing Module Testing

The preprocessing stage plays a crucial role in preparing the input data for the CNN model. Testing focuses on ensuring that the system correctly scales, normalizes, and resizes images to meet the requirements of the CNN architecture.

3. CNN Model Training and Testing

Training the CNN model is the heart of the project, and testing this process ensures its effectiveness in learning from the data. The system is evaluated for its ability to train on balanced datasets and produce accurate results without overfitting or underfitting.

4. Prediction Module Testing

The prediction module is tested to ensure the system accurately classifies new input images and handles invalid data gracefully. A variety of images, including those with distinct or borderline characteristics, are submitted to evaluate the model's reliability.

5. System Performance and Scalability

Performance testing evaluates the system's ability to handle computational demands during preprocessing, training, and prediction. Scalability testing ensures that the system can process large datasets efficiently, maintaining consistent performance without crashes or delays.

6. Security Testing

Security testing ensures that the system is robust against malicious activities and unauthorized access. This includes testing the sanitization of uploaded datasets to prevent injection of harmful scripts.

7. Database Testing

Database testing validates the system's ability to manage datasets, models, and predictions consistently. It ensures that the datasets are stored correctly and can be retrieved without errors.

CHAPTER 5

RESULTS

OUTPUT

```
PS C:\Users\SHIVA> & C:/Users/SHIVA/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/SHIVA/
E/C49FAAKZ/mini_project.py
Train Images Shape: (4217, 128, 128, 3)
Train Labels Shape: (4217,)
Training Data Shape: (3373, 49152)
Validation Data Shape: (844, 49152)
PCA-Reduced Training Data Shape: (3373, 50)
Validation Accuracy: 0.792654028436019
Classification Report:
                    precision    recall  f1-score   support

          cataract       0.80      0.72      0.76       233
diabetic_retinopathy       0.98      0.98      0.98       224
          glaucoma       0.69      0.63      0.66       188
            normal       0.69      0.82      0.75       199

          accuracy                           0.79       844
         macro avg       0.79      0.79      0.79       844
      weighted avg       0.80      0.79      0.79       844

Predicted class: cataract
PS C:\Users\SHIVA>
```

```
PS C:\Users\SHIVA> & C:/Users/SHIVA/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/SHIVA/Downloads/mini_project.py
Train Images Shape: (4217, 128, 128, 3)
Train Labels Shape: (4217,)
Training Data Shape: (3373, 49152)
Validation Data Shape: (844, 49152)
PCA-Reduced Training Data Shape: (3373, 50)
Validation Accuracy: 0.8056872037914692
Classification Report:
                    precision    recall  f1-score   support

          cataract       0.83      0.71      0.76       233
diabetic_retinopathy       0.99      0.97      0.98       224
          glaucoma       0.69      0.69      0.69       188
            normal       0.71      0.84      0.77       199

          accuracy                           0.81       844
         macro avg       0.80      0.80      0.80       844
      weighted avg       0.81      0.81      0.81       844

Predicted class: diabetic_retinopathy
PS C:\Users\SHIVA>
```

CHAPTER 6

CONCLUSIONS AND FUTURE SCOPE

PROJECT CONCLUSION

The "Eye Disease Classification and Prediction Using CNN" project illustrates the application of deep learning techniques in the field of medical imaging, specifically for the detection of eye diseases. Convolutional Neural Networks (CNNs) were utilized to analyze and classify eye images into distinct categories, including glaucoma, diabetic retinopathy, cataract, and normal conditions. The model was trained and evaluated on a dataset comprising medical eye images, achieving promising results in terms of accuracy and classification performance.

The study highlights the potential of CNN-based systems to assist healthcare professionals in the early diagnosis and management of eye diseases. By automating the detection process, such systems can save time, reduce diagnostic errors, and provide a cost-effective solution, especially in areas with limited access to ophthalmologists. The results underscore the importance of integrating advanced machine learning models into real-world healthcare workflows to enhance efficiency and accessibility.

Despite the success of the model, certain challenges were encountered, including class imbalance and the need for a larger, more diverse dataset to improve model generalization. Addressing these limitations through future research, such as incorporating data augmentation techniques and exploring transfer learning, could significantly enhance the model's performance. Additionally, deploying this model as a part of mobile or web applications could expand its usability and reach.The deployment of such a system in rural clinics and telemedicine platforms could bridge the gap in healthcare accessibility, particularly in regions where specialized medical expertise is scarce. Additionally, integrating this system with other diagnostic tools, such as optical coherence tomography (OCT) and fundus imaging devices, could provide a more comprehensive diagnostic framework for ophthalmologists.

FUTURE SCOPE

The "Eye Disease Classification and Prediction Using CNN" project has demonstrated promising results, paving the way for several advancements and applications in the field of medical imaging and healthcare technology. The future scope of this project includes:

1. **Enhanced Dataset Diversity:**
   - Expanding the dataset to include more diverse and rare eye diseases will improve the model's generalizability and accuracy.
   - Collaboration with medical institutions and research organizations can provide access to high-quality, annotated datasets.

2. **Integration with Real-World Systems:**
   - Deployment of the model in telemedicine platforms and mobile applications can enable remote diagnosis, making healthcare accessible to rural and underserved areas.
   - Integration with diagnostic devices like optical coherence tomography (OCT) scanners and fundus cameras can create a comprehensive diagnostic tool.

3. **Multi-Modal Learning:**
   - Combining image data with patient demographic information, medical history, and genetic data can enhance the predictive power of the model and provide more personalized insights.

4. **Improved Model Performance:**
   - Implementing transfer learning, pre-trained models, or ensemble learning techniques can enhance accuracy and efficiency.

5. **Real-Time Processing:**
   - Enhancing the model's speed and reducing latency to enable real-time predictions, which is crucial for point-of-care applications.

6. **Global Outreach:**
   - Scaling the system for multi-language support and region-specific disease variations to cater to global healthcare needs.
   - Training the model on data from diverse populations to reduce bias and improve its applicability worldwide.

7. **Education and Training:**
   - Using the model as a teaching tool for medical students and trainees to better understand the characteristics of eye diseases through AI-generated insights.

BIBILOGRAPHY

- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems, 25, 1097–1105.
    - Introduces CNNs and their application to image classification.
- Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv preprint arXiv:1409.1556.
    - Discusses VGG networks, useful for medical image processing.
- Gulshan, V., Peng, L., Coram, M., et al. (2016). *Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs*. JAMA, 316(22), 2402–2410.
    - Highlights the use of CNNs for eye disease detection.
- https://www.frontiersin.org/journals/computerscience/articles/10.3389/fcomp.2023.1252295/full
- https://www.researchgate.net/publication/372717102_Eye_Disease_Classification_Using_Deep_Learning_Techniques
- https://ieeexplore.ieee.org/document/10212139
- https://ijarsct.co.in/Paper4659.pdf
- https://github.com/somaiaahmed/Eye-diseases-classification