VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

IBTASAM UR REHMAN

MACHINE LEARNING APPROACHES FOR DETECTING OPHTHALMIC DISEASE

Major Computer Science
Major Code: 8480101

MASTER THESIS

HO CHI MINH CITY, JULY 2025

ĐẠI HỌC QUỐC GIA TP. HCM

TRƯỜNG ĐẠI HỌC BÁCH KHOA

IBTASAM UR REHMAN

PHƯƠNG PHÁP ỨNG DỤNG MÁY HỌC ĐỂ PHÁT HIỆN BỆNH NHÃN KHOA

Chuyên ngành: Khoa học máy tính

Mã số: 8480101

LUẬN VĂN THẠC SĨ

TP. HỒ CHÍ MINH, tháng 6 năm 2025

THIS THESIS IS COMPLETED AT HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY - VNU-HCM

Cán bộ hướng dẫn khoa học : PGS.TS Phạm Hoàng Anh

Cán bộ chấm nhận xét 1 : TS. Võ Đăng Khoa

Luận văn thạc sĩ được bảo vệ tại Trường Đại học Bách Khoa, ĐHQG Tp. HCM ngày 05 tháng 06 năm 2025

Thành phần Hội đồng đánh giá luận văn thạc sĩ gồm:

1. PGS.TS. Lê Hồng Trang
2. TS. Trần Tuấn Anh
3. TS. Võ Đăng Khoa
4. PGS.TS Phạm Hoàng Anh

Xác nhận của Chủ tịch Hội đồng đánh giá LV và Trưởng Khoa quản lý chuyên ngành sau khi luận văn đã được sửa chữa (nếu có).

CHỦ TỊCH HỘI ĐỒNG **TRƯỞNG KHOA.....**

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHIỆM VỤ LUẬN VĂN THẠC SĨ

Họ tên học viên: Rehman Ibtasam MSHV: 2370300

Ngày, tháng, năm sinh: 05/02/1998 Nơi sinh: Pakistan

Chuyên ngành: Khoa học máy tính Mã số:

I. TÊN ĐỀ TÀI: PHƯƠNG PHÁP ỨNG DỤNG MÁY HỌC ĐỂ PHÁT HIỆN BỆNH NHÃN KHOA

NHIỆM VỤ VÀ NỘI DUNG:

II. NGÀY GIAO NHIỆM VỤ: 13/01/2025

III. NGÀY HOÀN THÀNH NHIỆM VỤ: 04/06/2025

IV. CÁN BỘ HƯỚNG DẪN (Ghi rõ học hàm, học vị, họ, tên): PGS.TS Phạm Hoàng Anh

Tp. HCM, ngày 05 tháng 06 năm 2025

CÁN BỘ HƯỚNG DẪN
(Họ tên và chữ ký)

CHỦ NHIỆM BỘ MÔN ĐÀO TẠO
(Họ tên và chữ ký)

TRƯỞNG KHOA……
(Họ tên và chữ ký)

Ghi chú: Học viên phải đóng tờ nhiệm vụ này vào trang đầu tiên của tập thuyết minh LV

Acknowledgements

I express my sincere appreciation to my supervisor for their invaluable guidance and support throughout the development of this project. Their expertise and insights have been instrumental in shaping the direction and implementation of the application. I am grateful for the opportunity to work independently and for the encouragement to explore new ideas and approaches. I would also like to thank the open source community for their contributions to the technologies and libraries that formed the foundation of this project. The wealth of resources and collaborative spirit have been invaluable in overcoming challenges and achieving milestones. I extend my heartfelt thanks to my friends and family for their unwavering support and encouragement. Their belief in my abilities and understanding during the highs and lows of these endeavours have been a constant source of motivation.

Abstract

Cataracts are one of the leading causes of global vision impairment and require accessible and efficient diagnostic tools. In an effort to develop a cross-platform solution for people unable to afford or access diagnostic procedures, this study explores the application of machine learning for cataract detection using image classification. We developed a methodology involving image preprocessing, feature extraction, and classification using Support Vector Machines (SVM) with Linear, Polynomial, and Radial Basis Function (RBF) kernels as well as Random Forest classifiers. Image features including color (HSV mean), edge (Canny edge detection) and shape (Hu moments) were extracted and flattened into feature vectors. The models were trained and evaluated on a data set of 532 training images and 126 test images. Hyperparameter tuning was performed using GridSearchCV to optimize model performance. The results show that the tuned RBF SVM achieved the highest accuracy of 95.24% on the test set with a consistent 95% F1 score, outperforming the tuned Random Forest (94.44%) and the tuned polynomial SVM (92. 86%). Linear SVM exhibited the lowest accuracy (88.89%). Across all model variations, default and tuned, the classification of the user-provided image was accurate, identifying it as either Cataract or Normal. This study highlights the potential of machine learning, particularly tuned RBF SVM for accurate and efficient cataract detection, offering a promising avenue for developing accessible diagnostic tools.

Keywords: Support Vector Machine, Image Processing, Random Forest Classifier, Hyperparameter Tuning, GridSearchCV

Declaration

I, Ibtasam Ur Rehman, student of the Department of Computer Science and Engineering, HCMC University of Technology (HCMUT) confirm that this project Machine Learning Application Method for the Detection of Ophthalmic Diseases is my own work and all figures, tables, equations and attached code snippets in this report are original nor they have not been taken from any other person's work, except where the work of others have been explicitly acknowledged and quoted and referenced. I understand that if failing to do will be considered case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly. In any case of plagiarism, I stand by my actions and am to be responsible for it. Ho Chi Minh City University of Technology therefore is not responsible for any copyright infringements conducted within my research.

I give consent for my work to be made available more widely to members of HCMC University of Technology (HCMUT) and public with interest in teaching, learning and research.

I give consent to copy of my thesis being shared with future students as an example.

Ibtasam Ur Rehman
7th July 2025

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Vision, the major human sense, contributes an essential responsibility in every stage of our lives. People sometimes take vision for granted, but without vision, we have difficulty learning, walking, reading, and participating in numerous events of life. Vision impairment occurs when an eye condition affects visual functions; however, if left untreated, it shows serious consequences for the individuals throughout the life span. The number of various consequences can be reduced by timely access to eye care. According to the World Health Organization (WHO) for 2023, 2.2 billion people worldwide suffer from vision impairments. Ophthalmic condition that causes vision impairment and blindness resulting in cataract, refractive error, glaucoma, diabetic retinopathy, age-related macular degeneration. It is assumed that around the world 36% of people with distance vision impairment due to refractive error and 17% of the community with vision impairment resulting from cataracts. The decline in visual perception affects all people of all ages; however, visual impairment and blindness are older than 50 years.

The human eye, a marvel of biological engineering, stands as one of the most vital and delicate organs in our bodies. It enables us to perceive and interpret the world around us, transforming light into the rich tapestry of visual experience. Vision, arguably our most relied upon sense, serves as a primary conduit for acquiring information, shaping our understanding and interaction with the environment. The remarkable capacity of the human eye allows for the differentiation of approximately 10 million distinct colors, a testament to its intricate design. Structurally, the human eye approximates a sphere with a diameter of roughly 2.5 cm (or 83.07 cm in circumference, as you specified), a complex assembly of specialized components working in concert. These components include the iris, the colored part of the eye that controls the size of the pupil; the pupil, the central opening in the iris that regulates the amount of light entering

the eye; the cornea, the transparent front surface that provides initial focusing power; the lens, a flexible structure behind the iris that fine-tunes focus for near and far vision; the retina, the light-sensitive tissue at the back of the eye that converts light into neural signals; and the optic nerve, which transmits these neural signals to the brain. This intricate anatomy is illustrated in Figure 1.1. One common condition affecting the lens of the eye is cataract. Cataracts are characterized by the clouding of the lens, leading to blurred vision, faded colors, and halos around lights. This clouding obstructs the passage of light to the retina, significantly impairing visual acuity. The lens, located behind the iris, is normally clear, allowing for sharp focus. However, with age or due to other factors such as trauma, diabetes, or prolonged exposure to ultraviolet radiation, proteins within the lens can clump together, causing the lens to become opaque. The location of cataract formation within the lens is crucial, affecting the specific type and severity of visual impairment. The most common location is the nucleus (nuclear cataract) or the cortex (cortical cataract) of the lens.

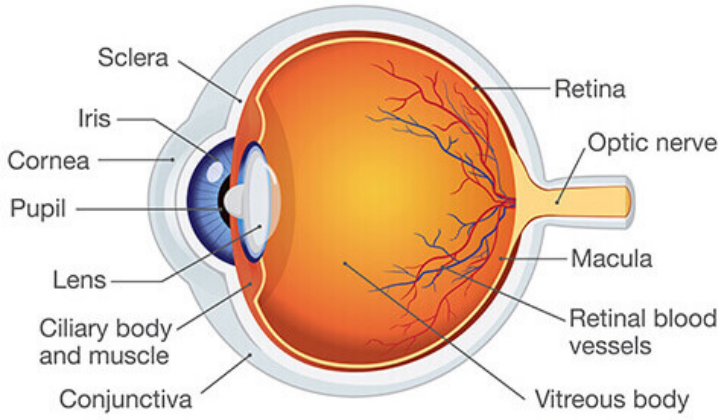

Figure 1.1: Diagram of the Human Eye Anatomy.

The cataract is a cloudy zone in the lens of the human eye. Cataracts are much more common when an individual reaches the age of 50 or older. At an early stage, someone may feel that they have cataracts because they develop slowly and steadily, but with time, cataracts can develop vision blurry or neutral however, as time passes, cataracts can lead to permanent vision loss. Initially, powerful lightning and corrective lenses can facilitate the treatment of cataracts. Indications of cataract include blurred vision, cloudy or dim vision, it can be sensitive to light, a problem when viewing at night, sudden changes in glasses or lens prescription, dual vision, faded colors, and halos near lights. The mentioned symptoms can fluctuate in terms of severity and may affect daily activities of existence. The root cause of cataracts includes aging, with certain conditions that are generally related to older individuals. Over time, proteins in the lens of the human eye can accumulate together, resulting in cloudiness and a visual perception problem. The continuous involvement to ultraviolet (UV) radiation from sunlight, specific medical con-

dition for example diabetics, hypertension, trauma injury associated with human eye, smoking, genetics can boost the risk of growing cataracts. Recognizing the symptoms and the reason for developing cataracts is important for the early detection and suitable treatment of the condition. Routine eye examination by healthcare professionals is essential for observing eye health and detecting cataracts in initial stages.

1.2 Motivation

The motivation behind my research derive from my amibition to handle the significant global impact of cataract which is a leading cause of vision impairment impacting billions of people worldwide. Recognizing the limitations of traditional methods which are often time consuming and costly diagnostic method. There are number of studies are being conducted but I didn't found any practical application that is utilizing machine learning algorithm to detect cataract. My aim is to utilize the power of machine learning algorithms within a cross platform mobile application. This approach seeks to empower peoples to perform preliminary cataract detection conveniently potentially leading to earlier diagnosis and intervention especially in underserved communities where access to specialized healthcare may be limited. Eventually this research is motivated by goal of improving global eye health and reducing the burden of cataract related vision impairment through innovative and readily available technological solution.

1.3 Problem statement

The typical approach for diagnosing cataract might be a considerable cause of vision impairment worldwide which may present due to several reasons:

- **High Prevalence and Impact:** Cataracts affect over 2.2 billion inidividuals globally and can lead to blurred vision, faded colors and halos around lights, significantly impacting the quality of life.
- **Reliance on Professional Consultation:** Traditional diagnosis typically demands consultations with healthcare professionals, which can be:
  - **Expert Dependency:** Diagnosis relies on experienced ophthalmologists, making it in accessible in regions with a shortage of specialists.
  - **Time-consuming:** Manual assessments require in person consultations, increasing the time needed for diagnosis and delaying early detection.

  - **Costly:** Specialized medical equipment such as slit lamps and fundus cameras are expensive and not widely available in resource-limited settings.
  - **Accessibility Issues:** Particularly in underserved or rural areas with limited access of ophthalmologists and diagnostic tools.

- **Need for Early and Accessible Detection:** Timely diagnosis is important for managing the condition and preventing potential vision loss. The lack of accessible and affordable diagnostic tools can delay detection and treatment.

To tackle with these challenges there is a need for innovative and efficient smart solutions that can:

- **Facilitate Early Detection:** Enabling individuals to assess the potential presence of cataracts in timely manner.
- **Improve Accessibility:** Delieverying a diagnostic tool that is readily available to wider population regardless of geographical location or access to specialized medical facilities.
- **Reduce the Burden on Healthcare Systems:** Maybe decreasing the demand for initial consultations for cataract screening.

1.4 Aims and objectives

The main goal of this research is to develop a cross platform mobile application for the efficient and accessible detection of cataract using machine learning algorithms applied to a dataset of labeled eye images. This aims to empower users with initial assessment capability without requiring immediate professional intervention.

Aims:

- To develop a user friendly cross platform application for the automated detection of cataract from eye images.
- To evaluate and compare the performance of Support Vector Machine (SVM) with various kernel functions (e.g. Linear, Polynomial and RBF) and the Random Forest classifier for cataract detection.
- To study the impact of hyperparameter tuning on the performance of both SVM and Random Forest classifiers for this specific task.

- To integrate the best performing machine learning model into the mobile application for real time cataract detection assessment.

Objectives:

- To implement Support Vector Machine (SVM) classifiers with Linear, Polynomial and Radial Basis Function (RBF) kernel for the binary classification of eye images (cataract vs normal) using their default parameter setting.
- To implement a Random Forest classifier for the binary classification of eye images (cataract vs normal) using its default parameter settings.
- To perform hyperparameter tuning for each of the implemented machine learning algorithms (SVM with all kernels and Random Forest) using appropriate techniques (e.g., GridSearchCV) to identify the optimal parameter configurations for cataract detection.
- To compare the performance of the SVM models (with default and tuned hyperparameters) and the Random Forest model (with default and tuned hyperparameters) using related evaluation metrics such as accuracy, precision, recall and F1 score on dedicated test dataset.
- To identify the machine learning model (algorithm and hyperparameter configuration) that yields the best performance for cataract detection based on the experimental results.
- To develop a cross platform mobile application that allows users to capture or upload eye images.
- To integrate the selected best performing machine learning model into the backend of the mobile application enabling it to process user provided eye image and provide a result comprising of detection status and confidence score to user.
- To design a user friendly interface for the mobile application which clearly define the assessment results to the user.
- To evaluate the overall functionality and performance of the developed application in providing accessible cataract risk assessment.

1.5 Proposed Solution

The proposed solution will follow a structured approach comprising of the development, evaluation of machine learning models for cataract detection and its integration into mobile application. The key stages of this approach are as follows:

- **Dataset Acquisition and Preparation:** Obtaining dataset of eye images labeled as 'cataract' and 'normal'. This dataset will be divided into training, testing sets.
- **Image Preprocessing:** Applying necessary image preprocessing techniques to the eye images, such as resizing, normalization and other enhancement methods to make sure consistent input for the machine learning models to improve their performance.
- **Model Implementation and Training (Default Parameters):** Implementing and training the following machine learning algorithms using their default parameter settings:
  - Support Vector Machine (SVM) with Linear, Polynomial and Radial Basis Function (RBF) kernel.
  - Random Forest classifier.
- **Hyperparameter Tuning:** Performing hyperparameter tuning for each of the implemented algorithm using techniques like GridSearchCV to find the optimal parameter configuration that maximize the performance of model for the cataract detection task.
- **Model Evaluation and Comparison:** Evaluating the performance of all models (with both default and tuned hyperparameters) on a test dataset using evaluation metrics (accuracy, precision, recall, F1-score, confusion matrix). The best performing model will be selected based on these results.
- **Mobile Application Development:** Developing a cross platform mobile application using a framework (e.g. Flutter) that allows users to capture or upload eye images.
- **Backend Integration:** Developing a backend infrastructure (e.g. using Flask) to host the selected machine learning model. This backend will receive eye images from the mobile application perform necessary preprocessing and use the trained model to predict the likelihood of cataract presence.
- **User Interface Development:** Designing and implementing a user friendly interface in the mobile application to display the cataract risk assessment results to the user in a clear and understandable manner.

1.6 Summary of contributions and achievements

This research aims to provide the accessible tool for ophthalmic assessment through mobile application for intial cataract detection using machine learning. Contributions and achievements are as follows:

- **Mobile Cataract Detection Application:** Development of a cross platform application for initial cataract assessment via smartphones.
- **SVM and Random Forest Comparison:** Performance analysis of SVM (all kernels) and Random Forest with default and tuned hyperparameters.
- **Optimal Model Identification:** Identification of the best machine learning model for accurate cataract detection from eye images.

Chapter 2

Literature Review

This section examines the application of machine learning and deep learning techniques in cataract classification. In recent decades both approaches have played a important role in assisting healthcare professionals with cataract detection diagnosis. These models have demonstrated remarkable performance in analyzing ophthalmic images.

2.1 Traditional Methods for Cataract Detection

Cataract detection traditionally depend upon the clinical examination techniques which are conducted by eye care professionals. The most common technique is the visual acuity test where patients asked to read letters on Snellen chart to examine their clarity of vision. Furthur fundamental technique is slit-lamp examination which uses specialized type of microscope and luminous light to inspect the anterior structure of the eye which include lens for any opacities. Retinal examination are often performed after pupil dilation which enables direct visualization of lens and retina to identify cataract and can help to detect other underlying conditions. Although tonometry is primarily used to measure intraocular pressure for glaucoma detection but it is often included in comprehensive eye exams where cataracts are suspected. Furthurmore contrast sensitivity testing is also used to evaluate how well patient can distinguish between various shades of light and dark a function often impaired by cataracts. These traditional techniques are non invasive but can widely used in clinical settings and they can provide reliable basis for diagnosing cataract.

2.2 Machine Learning Approaches

Agarwal et al.[1] developed a smartphone-based Android application for cataract detection, aiming to reduce time and costs compared to conventional clinical methods. The application incorporates machine learning algorithms, including KNN, SVM, and Naïve Bayes, along with image processing techniques using the OpenCV library. Users can either upload eye images from their gallery or capture new ones, which are then analyzed against pre-trained datasets to identify cataracts. The methodology involves several stages, such as image collection, data preprocessing with the Orange Tool, classification using KNN, and validation under ophthalmologists' supervision. Among the evaluated models, KNN demonstrated the highest accuracy of 83.07%, surpassing SVM and Naïve Bayes.

Behera et al.[2] proposed a structured model for detecting nuclear cataracts using fundus retinal images. The model processes images to generate binary representations highlighting blood vessels, which serve as the feature matrix for a Support Vector Machine (SVM) classifier. Various SVM kernels were evaluated, with the Radial Basis Function (RBF) achieving the highest precision of 95.2%. The study concluded that RBF-based SVM is effective for real-time detection and suggested further exploration of Convolutional Neural Networks (CNNs) to enhance performance.

Fuadah et al.[4] developed a mobile application for cataract detection using statistical texture analysis and K-Nearest Neighbor (k-NN) classification. The study addresses the challenges faced in rural areas of Indonesia, where limited access to ophthalmologists and diagnostic tools has made cataracts the leading cause of blindness. A dataset of 160 eye images, consisting of 80 normal and 80 cataract cases, was analyzed using preprocessing techniques such as cropping and grayscale conversion, followed by feature extraction using the gray level co-occurrence matrix (GLCM). The results achieved a precision of 97.5%, highlighting key features such as distinction, contrast, and uniformity, with the optimal k value for k-NN determined to be 1. The study concludes that this approach effectively differentiates between normal and cataract-affected eyes, offering a promising mobile solution for underserved communities.

Gao at al.[5] aimed to improve cataract detection and classification using artificial intelligence applied to fundus images. They collected 1,340 fundus images and developed a dual-stream cataract evaluation network (DCEN) to classify and grade cataracts. The DCEN model was trained and tested using deep learning algorithms, primarily ResNet models. The results demonstrated high accuracy, sensitivity, F1 score, and kappa coefficient across both cataract

classification and severity grading tasks, achieving an overall accuracy of 97.62

Vasan et al.[16] introduced two algorithms for analyzing eye images to assess cataract severity. The first algorithm utilizes feature extraction and histogram evaluation to classify eyes as healthy or affected by varying degrees of cataract based on mean intensity. The second method calculates the cataract area relative to the pupil using contour detection and Hough transforms. While the feature extraction method demonstrated greater versatility, the area-based approach required specific conditions for accurate results. The study emphasizes the importance of automation and the development of mobile applications to enhance accessibility to cataract detection.

Junayed et al.[7] explored cataract detection using deep learning with fundus images. They utilized multiple datasets, including HRF, FIRE, ACHIKO-I, and IDRiD, for training and testing purposes. The study introduced a convolutional neural network (CNN)-based model named CataractNet for cataract detection. The performance of CataractNet was compared with five pretrained CNN models under different dataset-splitting conditions. Results showed that CataractNet outperformed the other models in terms of accuracy and evaluation metrics while also exhibiting lower time complexity. The study highlighted CataractNet's effectiveness across multiple metrics, including accuracy, precision, recall, specificity, MCC, and F1-score.

Khalaf and Abdulateef[8] conducted an experiment to classify ophthalmic diseases using YOLOv8. They sourced fundus images from platforms such as Roboflow, Kaggle, and medical clinics, standardizing them to 224×224 pixels. A total of 5,887 images were collected and split into training and testing datasets. YOLOv8n-cls (nano) was selected for its efficiency, compact size, speed, and performance. The model was trained using specific parameters, including the ADAM optimizer, learning rate, batch size, weight decay, and 30 epochs. Several evaluation metrics, such as accuracy, precision, recall, and F1-score, were used to assess its performance. The proposed model achieved a 94% accuracy rate in classifying various eye diseases, surpassing traditional CNN models.

Meegada et al.[9] proposed an efficient cataract detection approach integrating deep learning and machine learning models. Feature extraction was performed using a Convolutional Neural Network (CNN), while classification was carried out using various ML and DL models, including Support Vector Machine (SVM), Random Forest (RF), and Deep Neural Networks (DNN). The experimental process involved multiple stages, such as data preparation, augmentation, model selection, training, and optimization. The proposed method achieved a high detection accuracy of 98.4

2.2. Machine Learning Approaches

Rana and Galib[13] introduced a mobile-based cataract detection solution utilizing a smartphone camera. The application identifies cataracts by analyzing pupil images through color detection and classification. This system is particularly beneficial for rural areas with limited access to medical professionals and specialized diagnostic tools like slit-lamp cameras. The method leverages OpenCV, SDK, and NDK for image processing, offering an affordable and accessible solution for early cataract detection. Initial experiments demonstrated a 90% accuracy rate in detecting cataracts at various stages.

Ranjan et al.[14] developed a mobile application for cataract detection using deep learning techniques. The proposed system demonstrated an accuracy of 98.79% while also maintaining high sensitivity and specificity. The detection algorithm is based on a CNN model consisting of three key layers: convolution, pooling, and fully connected layers, which facilitate pattern recognition. Users can capture eye images with their mobile cameras for analysis, and if a cataract is detected, the system provides an alert along with the severity level of the condition.

Yang et al.[18] carried out an experiment using 504 sets of images, labeled by experienced ophthalmologists, for cataract detection and classification. They employed a backpropagation neural network for classification while extracting multiple features, including luminance features to assess retinal image clarity, gray co-occurrence matrix features that provide information such as entropy, contrast, and inverse details, and gray-gradient co-occurrence matrix features that capture gray level and gradient details. The model's performance was evaluated using the Receiver Operating Characteristic (ROC) curve and a confusion matrix, illustrating the relationship between true positive and false positive rates.

2.2.1 Ensemble Methods for Cataract Classification

Ensemble methods in machine learning blend the predictions of multiple models into one prediction to improve classification performance. These methods can substantially improve the accuracy and robustness of cataract classification especially when handling with complex datasets such as medical images.

2.2.2 Types of Ensemble Methods

Bagging

Bagging Aggregating comprises of training multiple models on different sections of the data and combining their predictions into one. One of them most common example is the Random Forest algorithm which uses multiple decision trees to classify cataract severity.

Boosting

Boosting technique build models step by step where each new model aims to correct errors made by the last ones. Algorithms like AdaBoost or XGBoost are widely used to improve cataract classification by focusing on hard to classify cases.

Stacking

Stacking involves training multiple models and combining their predictions using a meta model. This approach may utilize the capacities of numerous classifiers to improve accuracy in cataract detection.

2.2.3 Benefits of Ensemble Methods

Ensemble methods offer many advantages in cataract classification as listed below.

- **Improved Accuracy:** By combining numbers of models ensemble methods may provide better results than single model.
- **Robustness:** Ensembles are less tends to overfitting and they can handle noisy data more effectively and efficiently.
- **Better Generalization:** Ensembles tends to generalize better specifically in real world application where data can vary.

2.3 Deep Learning Approaches

Ramakrishnan et al.[12] perform a study suggesting hybrid deep learning model for cataract detection from specialized fundus images. They develop CNN SVM architecture for feature extraction and evaluate four transfer learning model (e.g. MobileNet, VGG 19, ResNet 50, Inception V3) for classification and optimized using Intel's oneDNN library in oneAPI environment. This method achieves 98.36% accuracy which is considers as best with MobileNet on the ODIR dataset outperforming existing approaches. They implemented image preprocessing techniques which include resizing, pixel normalization and histogram equalization for contrast enhancement. The hybrid CNN SVM model combines CNN hierarchical feature extraction with SVM discriminative power addressing challenges like noise sensitivity and overlapping features in medical images. This computationally efficient framework illustrate promise for real world clinical deployment in early cataract diagnosis.

Dong et al.[3] propose deep learning based method for cataract detection along with its severity classification utilzing retinal fundus images. This method employ a 5 layer convolutional neural network (CNN) built on Caffe framework to get discriminative features and obtaining performance compared to traditional techniques. This study utilizes 7851 clinically labeled fundus images (e.g. 4671 normal, 2176 mild, 622 medium and 382 severe cases) applies maximum entropy transformation for preprocessing to enhance image contrast while maintainnig retinal structures. Extracted features from the CNN's fully connected (fc5) layer are classified using Softmax and SVM, with results compared against conventional wavelet-based retinal vascular features. Key findings demonstrate that CNN features with Softmax classification yield the highest accuracy:

- 90.82% for 4-class grading (normal, mild, medium, severe)
- 94.07% for binary classification (normal vs. cataract)

This outperforms prior methods relying on wavelet features (84.7% accuracy) and manual feature engineering. The work underscores the potential of deep learning in medical image analysis, though computational demands remain a limitation. Future directions include model optimization and integration with advanced AI frameworks.

2.3.1 Transfer Learning Techniques

Transfer learning is a beneficial technique for cataract detection especially with limited data because it involve adapting pretrained models on large datasets to this specific task saving training time and improving performance.

2.3.2 Pretrained Convolutional Neural Network (CNN) Architectures

A wide range of pretrained CNN architecture has been successfully employed the foundation for transfer learning in cataract detection. These models consists of hierarchical feature representations from millions of images which may provide strong starting point for analyzing images. Some of the most commonly utilized architectures are follows:

- **VGGNet (VGG16, VGG19):** These are deep convolutional neural networks which are known for their simplicity and effectiveness in the context of image classification which are making them suitable for cataract detection. These models with multiple layers respectively use small 3x3 convolution filters to gather fine grained features which are important for detecting subtle changes in eye image which are caused by cataracts. By applying transfer learning with pretrained weights from ImageNet VGG models can be finetuned on cataract datasets like slitlamp or fundus images to classify cataracts with its stages (e.g. no cataract, mild, moderate, severe). Apart from simplicity VGG models are computationally expensive due to their large number of parameters, which can lead to overfitting on small datasets. However, ability to learn deep features from image and makes them useful tool for automated cataract classification.

- **ResNet (ResNet50, ResNet101):** These models can be utilize for cataract classification due to their deep architecture and residual connections which is helpful to kern more about complex features form eye images. Fine tuning of these models on cataract dataset either fundus images or slit-lamp images allows these models to classify there severity. By utilizing transfer learning ResNet models require less data and training time and providing high accuracy in detecting cataracts but they can be computationally intensive.

- **Inception (InceptionV3):** The Inception model is deep convolutional neural networks which is well known due to its efficiency in tackling complex image classification tasks which is making a strong choice for cataract detection. These model use unique architecture Inception components which combine numerous filter sized at each layer which allows it to gather both fine and coarse features of images. In the context of cataract classification, InceptionV3 can be fine tuning on cataract dataset. Its ability to learn various set of features which can be combined with transfer learning from pretrained weights and

enables accurate detection even with limited data. This model is computationally efficient which can provide high accuracy for cataract classification

- **MobileNet (MobileNetV2, MobileNetV3):** These models are lightweight convolutional neural networks which are designed for efficient performance on mobile devices which maki them ideal for cataract classificaition in resource constrained settings. These models use depth wise separable convolutions and optimized architecture which is use to reduce computational load while maintaining the accuracy. In the context of cataract detection MobileNet models can be fine tuned on eye dataset to classify the presence and severity of cataracts. The small size of these models and fast inference make them suitable for real time screening applications.

- **DenseNet (DenseNet121, DenseNet201):** These models are effective for cataract classification due to their compact connectivity where every layer gets input from all previous layers. This promotes feature reuse and strengthens the gradient flow wich is allowing the model to handle detailed patterns in eye image. If we implement fine tuned on cataract datasets DenseNet can help to classify different stages of cataracts with high accuracy. Its compact and efficient structure also helps reduce overfitting making it well for medical image analysis when having limited data.

- **DenseNet (DenseNet121, DenseNet201):** EfficientNet is highly optimized convolutional neural network which have a capability to balance accuracy and computational efficiency by scaling depth, width and resolution simultaneously. In the context of cataract detection EfficientNet can be fine tuned on eye images to accurately classify cataract stages while using less parameters and less computation compared to other traditional models.

2.4 Why Machine Learning Algorithm is used instead of deep learning ?

For cataract classification machine learning (ML) algorithms like Support Vector Machines (SVM) and Random Forests (RF) are preferred over deep learning (DL) models particularly when dealing with limited datasets and resource constraints. This preference stems from several key factors:

2.4.1 Dataset Constraints

The study utilizes relatively small dataset (532 training images) for cataract classification. Deep Learning (DL) models such as Convolutional Neural Networks (CNNs) typically necessitate large scale datasets (ranging from thousands to millions of images) to effectively learn intricate features, generalize well to unseen data and mitigate overfitting. In contrast traditional ML models can efficiently leverage handcrafted features specific to cataract detection. These features such as color e.g. HSV mean), edge patterns (e.g., Canny edges), and shape (e.g., Hu moments) are directly identifiable and do not require the extensive data volumes that DL models need to automatically learn representations.

2.4.2 Computational Efficiency

Deep learning models demand high computational resources including powerful GPUs or TPUs for both training and inference. The study objective to develop a lightweight solution deployable on mobile devices (via Flutter) makes traditional ML models a more suitable choice due to their inherent computational efficiency. SVM and RF models train considerably faster on smaller datasets compared to DL models which would require significantly longer training times and extensive optimization efforts. This efficiency is crucial for real world applications where rapid deployment and operation on resource-limited platforms are paramount.

Chapter 3

Methodology

This chapter details the methodology utilized in this result for the cataract detection task. As our methodology is divided into multiple steps inclusive data acquisition, preprocessing, feature extraction, model training and evaluation. The process utilizes image processing approches and machine learning algorithms to classify eye images into 'Normal' and 'Cataract' classes.

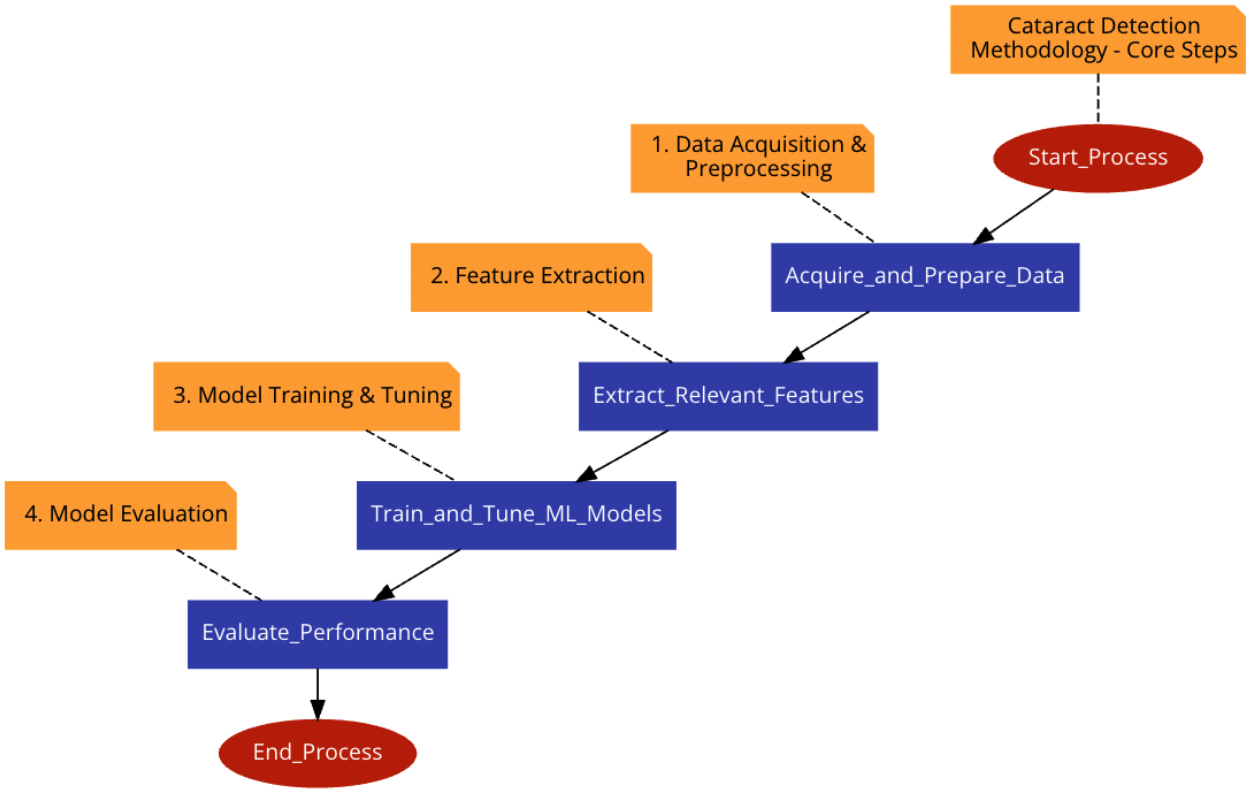

Figure 3.1: Work flow of the proposed method

3.1 Data Acquisition and Preprocessing

The primary step include acquiring and preparing the image for model training.

3.1.1 Data Sources

The dataset is gatherd form onlie source (e.g. Kaggle Dataset) and used for this study which consists of collection of eye images categorized into two distinct classes: 'Normal' and 'Cataract'. The dataset are organized into dataset/train and dataset/test directories indicating structured collection of images.

3.1.2 Image Loading and Resizing

The `load_images_from_folder` function is responsible for handling the data and initial preprocessing of the images. For every image file found with in the 'normal' and 'cataract' directories of specified folder `TRAIN_FOLDER` or `TEST_FOLDER` the following steps are performed:

```
image = cv2.imread(img_path)
if image is None:
    continue
image = cv2.resize(image, img_size)
```

The `cv2.imread()` function from OpenCV is used to load image from specific file path (`img_path`) into NumPy array. A check for `None` make sure that any unreadable files are skipped. Then `cv2.resize()` resizes the loaded image to fixed dimension of 100x100 pixels as defined by `img_size = (100, 100)`. This standardization of image size is important to make sure a consistent input dimension for machine learning model.

3.1.3 Image Conversion

Grayscale Conversion (for edge and shape features)

For the extraction of edge and shape features, the RGB images are converted to grayscale. This conversion simplifies the image data by reducing it to a single channel representing pixel intensity, which is necessary for edge detection and shape analysis. The grayscale intensity Y of a pixel with RGB values (R,G,B) is typically calculated using a weighted average:

$$Y = w_R \cdot R + w_G \cdot G + w_B \cdot B$$

where $w_R \approx 0.299$, $w_G \approx 0.587$, and $w_B \approx 0.114$ are the weights assigned to the red, green, and blue channels, respectively, reflecting human perception of color luminance.

```
    for img in images:
        img = img.reshape(100, 100, 3)
        img_uint8 = (img * 255).astype(np.uint8)
        gray = cv2.cvtColor(img_uint8, cv2.COLOR_RGB2GRAY)
```

HSV Color Space Conversion (for color features)

To facilitate the extraction of color-based features, the images are converted from the RGB color space to the HSV (Hue, Saturation, Value) color space. This conversion separates color information from brightness, making the color features more robust to lighting variations. The conversion process from normalized RGB values $(R,G,B) \in [0,1]$ to HSV values $(H \in [0°, 360°), S \in [0,1], V \in [0,1])$ can be mathematically represented as follows:

The Value (V) is given by:

$$V = \max(R, G, B)$$

The Saturation (S) is calculated as:

$$S = \begin{cases} 0 & \text{if } V = 0 \\ \frac{V - \min(R,G,B)}{V} & \text{otherwise} \end{cases}$$

The Hue (H) is determined by: Let $\Delta = V - \min(R, G, B)$.

$$H = \begin{cases} 0^\circ & \text{if } \Delta = 0 \\ 60^\circ \cdot \left(\frac{(G-B)}{\Delta} \mod 6 \right) & \text{if } V = R \\ 60^\circ \cdot \left(\frac{(B-R)}{\Delta} + 2 \right) & \text{if } V = G \\ 60^\circ \cdot \left(\frac{(R-G)}{\Delta} + 4 \right) & \text{if } V = B \end{cases}$$

The Hue value is typically normalized to the range $[0, 1]$ by dividing by 360°.

```
    for img in images:
        img = img.reshape(100, 100, 3)
        img_uint8 = (img * 255).astype(np.uint8)
        hsv = cv2.cvtColor(img_uint8, cv2.COLOR_RGB2HSV)
        h_mean, s_mean, v_mean = np.mean(hsv, axis=(0, 1))
        color_features.append([h_mean, s_mean, v_mean])
    color_features = np.array(color_features)
```

3.1.4 Image Enhancement

Unsharp Masking

An unsharp masking technique is applied to enhance the edges and details of the images. This process involves subtracting a blurred version of the image from the original, resulting in sharpened edges and improved clarity. This enhancement step aims to improve the visibility of subtle differences between normal and cataract-affected eyes, thereby aiding the model in accurate classification.

Mathematically, let $I(x, y)$ represent the original image, where (x, y) are the pixel coordinates. The blurred version of the image, $I_{blurred}(x, y)$, is typically obtained by convolving the original image with a low-pass filter, such as a Gaussian kernel $G(x, y, \sigma)$:

$$I_{blurred}(x,y) = I(x,y) * G(x,y,\sigma)$$

where $*$ denotes the convolution operation and σ is the standard deviation controlling the blur amount.

The unsharp mask or detail layer, $D(x,y)$, is then calculated as the difference between the original and the blurred image:

$$D(x,y) = I(x,y) - I_{blurred}(x,y)$$

The enhanced image, $I_{enhanced}(x,y)$, is obtained by adding a scaled version of this detail layer back to the original image:

$$I_{enhanced}(x,y) = I(x,y) + k \cdot D(x,y)$$

where k is a positive gain factor controlling the sharpening intensity. Substituting the expression for $D(x,y)$, we get:

$$I_{enhanced}(x,y) = I(x,y) + k \cdot (I(x,y) - I_{blurred}(x,y))$$
$$I_{enhanced}(x,y) = (1+k) \cdot I(x,y) - k \cdot I_{blurred}(x,y)$$

This equation describes the unsharp masking process, where the enhanced image is a linear combination of the original and its blurred version, with the detail information amplified by the factor k. The following Python code snippet demonstrates function of unsharp masking technique as follows:

```
1 def unsharp_mask(image, sigma=1.0, strength=1.5):
2         blurred = cv2.GaussianBlur(image, (0, 0), sigma)
3         sharpened = cv2.addWeighted(image, 1 + strength, blurred,
             -strength, 0)
4     return sharpened
```

3.1.5 Data Splitting (Training, Validation, Testing)

The methodology loads image dataset from specified folder (e.g. dataset/train and dataset/test). This implies predefined split of data into training and testing sets. The training set `(X_train, y_train)` is used to train the model however the testing set `(X_test, y_test)` is used to evaluate the performance of model on unseen data. A validation set is used during hyperparameter tuning to evaluate the model's performance on data not seen during training which is helpful to prevent overfitting to the training data.

The pixel value of the loaded images are normalized by dividing by 255.0:

```
image = image / 255.0
```

This scaling make sue that the pixel value are in the range [0, 1] which can improve the training stability and convergence of the algorithm. The processed images and their corresponding labels (0 for 'normal', 1 for 'cataract') are stored in NumPy arrays:

```
images.append(image)
labels.append(0 if label.lower() == 'normal' else 1)
return np.array(images), np.array(labels)
```

The training data `X_train` and testing data `X_test` are reshaped to a flat vector for use with the Support Vector Machine (SVM) and Random Forest models:

```
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
```

For image of size 100x100 with 3 color channels this results in a feature vector of 100×100×3=30000 dimensions.

3.2 Feature Extraction

This section details the methods used to extract relevant features from the preprocessed images, which are essential for the accurate classification of cataract and normal eye images.

3.2.1 Edge Features - Canny Edge Detection

Edge features are extracted using the Canny edge detection algorithm. This algorithm is used to detect edges in the grayscale images, highlighting the boundaries of objects. The Canny edge detector involves several steps, including Gaussian filtering to reduce noise, gradient calculation (e.g., using Sobel operators), non-maximum suppression to thin edges, and hysteresis thresholding to finalize edge detection. The output is a binary edge map E, where $E(x,y) = 1$ if a pixel at (x,y) is an edge, and $E(x,y) = 0$ otherwise.

3.2.2 Edge Pixel Count

After applying the Canny edge detection, the number of edge pixels is counted. This count, denoted as N_{edge}, is calculated as:

$$N_{edge} = \sum_{x=1}^{W} \sum_{y=1}^{H} E(x,y)$$

where W and H are the width and height of the image, respectively. This count serves as a feature representing the overall amount of edge information in the image.

```
1        edges = cv2.Canny(img_uint8, 100, 200)
2        edge_count = np.sum(edges > 0)
3        edge_features.append(edge_count)
```

3.2.3 Color Features - HSV Mean Calculation

Color features are extracted by converting the images to the HSV (Hue, Saturation, Value) color space. This color space is chosen because it separates color information from brightness, making it more robust to variations in lighting. For an image I_{HSV} with Hue (H), Saturation (S), and Value (V) channels, the mean values are calculated as:

$$\bar{H} = \frac{1}{N}\sum_{i=1}^{N} H_i, \quad \bar{S} = \frac{1}{N}\sum_{i=1}^{N} S_i, \quad \bar{V} = \frac{1}{N}\sum_{i=1}^{N} V_i$$

where N is the total number of pixels in the image, and H_i, S_i, V_i are the Hue, Saturation, and Value of the i-th pixel, respectively. The mean values $(\bar{H}, \bar{S}, \bar{V})$ provide a concise representation of the overall color distribution in the image and are used as color features.

```
1    hsv = cv2.cvtColor(img_uint8, cv2.COLOR_RGB2HSV)
2    h_mean, s_mean, v_mean = np.mean(hsv, axis=(0, 1))
3    color_features.append([h_mean, s_mean, v_mean])
```

3.2.4 Shape Features - Hu Moments Extraction

Shape features are extracted using Hu moments, which are a set of seven values $(\phi_1, \phi_2, ..., \phi_7)$ that are invariant to image translation, rotation, and scale changes. These moments are derived from the image's central moments (μ_{pq}), which are defined as:

$$\mu_{pq} = \sum_{x=1}^{W}\sum_{y=1}^{H}(x-\bar{x})^p(y-\bar{y})^q I(x,y)$$

where $\bar{x}$ and $\bar{y}$ are the centroid of the image, and $I(x,y)$ is the intensity of the pixel at (x,y). The seven Hu moments are specific combinations of these central moments, designed to achieve invariance. For example, the first Hu moment ϕ_1 is given by:

$$\phi_1 = \eta_{20} + \eta_{02}$$

where η_{pq} are the normalized central moments. The remaining six Hu moments are more complex combinations of η_{pq}. These seven Hu moments provide a compact representation of the image's shape.

```
        gray = cv2.cvtColor(img_uint8, cv2.COLOR_RGB2GRAY)
        moments = cv2.moments(gray)
        hu_moments = cv2.HuMoments(moments).flatten()
        shape_features.append(hu_moments)
```

3.2.5 Color Feature Visualization

To understand the distribution and separability of color features, visualizations are created. Scatter plots are generated to show the distribution of the mean Hue ($\bar{H}$), Saturation ($\bar{S}$), and Value ($\bar{V}$) values for normal and cataract affected images.

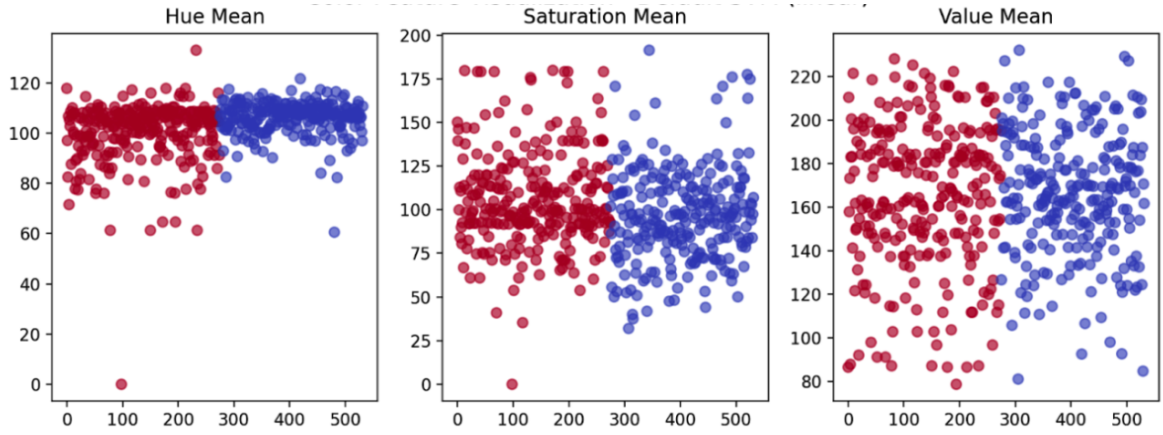

Figure 3.2: Mean HSV Color Features Visualization.

3.2.6 Edge Pixel Feature Visualization

The edge pixel counts (N_{edge}) are visualized using scatter plots to observe their distribution across normal and cataract affected images.

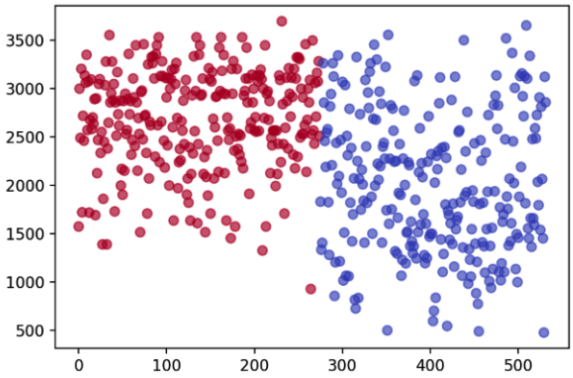

Figure 3.3: Edge Pixel Feature Visualization

3.2.7 Shape Feature Visualization

The Hu moments ($\phi_1, ..., \phi_7$) are visualized to observe their distribution and separability specifically for cataract and normal eye images. Scatter plots are created to show the distribution of each Hu moment value for these two distinct image categories. These visualizations provide a assessment of how distinct the shape characteristics are between cataract and normal eye images.

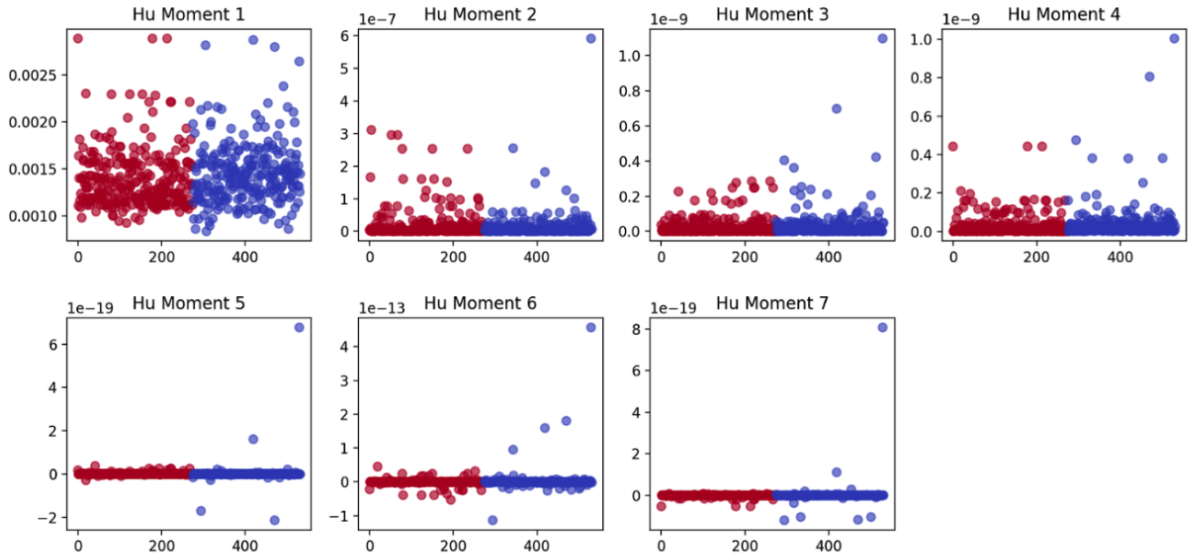

Figure 3.4: Shape Feature Visualization

3.2.8 Feature Vector Flattening

Finally, the extracted features (HSV mean values $[\bar{H}, \bar{S}, \bar{V}]$, edge pixel count $[N_{edge}]$, and Hu moments $[\phi_1, \phi_2, ..., \phi_7]$) are flattened into a one-dimensional feature vector $\mathbf{f}$:

$$\mathbf{f} = [\bar{H}, \bar{S}, \bar{V}, N_{edge}, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7]^T$$

This flattened vector serves as the input to the machine learning models. Flattening the feature vectors ensures that the models receive the data in a format they can process, allowing for efficient training and prediction.

3.3 Model Training

This section details the training procedures employed for the machine learning models used in cataract detection, encompassing Support Vector Machines (SVM) and Random Forest classifiers. We explore both default parameter configurations and hyperparameter-tuned models to optimize performance.

3.3.1 Support Vector Machine

Support Vector Machines (SVMs) are robust supervised learning models that excel in classification tasks by constructing an optimal hyperplane that maximizes the margin between different classes. We trained SVMs with various kernel functions and meticulously tuned hyperparameters to ensure optimal performance.

1. **Regularization parameter** C controls the trade-off between maximizing the margin and minimizing the classification error. The objective function for SVM can be formulated as (3.3.1):

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) \tag{3.3.1}$$

   Here, $\mathbf{w}$ represents the weight vector, b is the bias, y_i are the target labels, and $\mathbf{x}_i$ are the feature vectors. A lower value of C results in a wider margin at the cost of misclassifications, while a higher value C aims to minimize misclassifications by potentially reducing the margin.

2. **Kernel Coefficient** γ determines the influence of a single training example. The decision boundary becomes more complex with larger values of γ. The RBF kernel is given by (3.3.2):

$$K(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x}-\mathbf{x}'\|^2} \tag{3.3.2}$$

   A low value of γ results in a smooth decision boundary, while a high value allows for more complex decision boundaries.

3. **Degree parameter** is relevant for the Polynomial kernel, determining the degree of the polynomial used in the kernel function as (3.3.3):

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^{\text{degree}} \tag{3.3.3}$$

   Here, c is a constant that trades off the influence of higher-order terms.

4. **coef0** parameter influences the trade-off between the linear and polynomial terms in the kernel. This is particularly relevant in the polynomial kernel, affecting the flexibility of the model.

3.3.2 GridSearchCV

This is a techinique which is widely used method in machine learning for hyperparameter tuning. It step by step searches through predefined set of hyperparameters which are evaluating each combination using cross validation to determine the best performing parameter configuration. This approach make sure that the model parameters are optimized not just for the training set but also for generalization to unseen data. By the tuning process GridSearchCV helps improve model accuracy, reduce overfitting,and incrase chance of performance. However in this experiment we have applied GridSearchSV to fine tune the paramter of Support Vector Machines and Random forest classifier.

3.3.3 Kernel Selection

As we study the performance of the Support Vector Machine (SVM) algorithm with three various kernel functions e.g. Linear, Polynomial and Radial Basis Function (RBF). The choice of kernel function is important in SVM as it decide the nature of decision boundary the model can learn.

- **Linear Kernel:** The linear kernel is simple kernel as it is suitable for linearly separable data. It compute the dot product of the input vectors:

$$K(x_i, x_j) = x_i^T x_j$$

  A linear kernel goal to find linear hyperplane that best separate the data points of different classes (e.g. Cataracts, Normal). It is computationally efficient and often perform well when the number of features is large compared to the number of samples. In our case as with flattened image vectors (30,000 features) a linear kernel might provide a good baseline and might be efficient to train the model.

- **Polynomial Kernel:** The polynomial kernel allows for learning the non linear decision boundaries which is done by mapping the input data to a higher dimensional space. The kernel function is defined as:

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$$

  where γ is kernel coefficient, r is the independent term and d is the degree of the polynomial. By different the degree (d) the model can fit more complex relationships in the data. We experimented with degrees 2, 3 and 4 during hyperparameter tuning to find ppropriate level of non linearity for Catract classification.

- **Radial Basis Function (RBF) Kernel:** The RBF kernel is widely used kernel that can model highly non linear decision boundaries. The Gaussian RBF kernel is defined as:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

  where γ is positive kernel coefficient. The RBF kernel map the input space into infinitely dimensional space allowing for very flexible decision boundaries. The γ parameter plays important role in the model complexity because a small γ leads to smoother decision boundary while a large γ can result in more complex and potentially overfit model and as explored multiple values of γ ('scale' and 'auto' as implemented in scikit-learn) during hyperparameter tuning.

```
    def train_svm_default(X_train, y_train, kernel_type):
    param_dict = {
        'linear': {'C': 1.0},
        'poly': {'C': 1.0, 'degree': 3, 'gamma': 'scale'},
        'rbf': {'C': 1.0, 'gamma': 'scale'}
    }
    params = param_dict[kernel_type]
    model = SVC(kernel=kernel_type, probability=True, **params)
    model.fit(X_train, y_train)
    extract_and_visualize_features(X_train, y_train, f"Default SVM
        ({kernel_type})")

    return model, params
```

3.3.4 Hyperparameter Tuned SVM Training - GridSearchCV

To optimize the SVM models and mitigate potential issues such as overfitting or underfitting, hyperparameter tuning was performed using GridSearchCV. This method systematically searches through a predefined grid of hyperparameter values, employing cross-validation to evaluate each combination and identify the best configuration that maximizes the model's performance on unseen data.

Linear Kernel

For the linear SVM, the regularization parameter C was tuned. A grid of values, such as 0.01, 0.1, 1, 10, and 100, was used to find the optimal C value. This range allows for exploration of both strong and weak regularization effects, ensuring the model generalizes well to new data.

Polynomial Kernel

For the polynomial SVM, the regularization parameter C, degree, and gamma were tuned. The grid search included values for C (same range as linear SVM), degrees of 2, 3, and 4 (to explore different polynomial complexities), and gamma values of 'scale' and 'auto' (to compare adaptive and fixed gamma settings).

RBF Kernel

For the RBF SVM, the regularization parameter C and gamma were tuned. The grid search included values for C (same range as linear SVM) and gamma values of 'scale' and 'auto'. Tuning these parameters is crucial for controlling the model's complexity and ability to generalize.

```
def train_hyperparameter_tuned_svm(X_train, y_train, kernel_type):
    param_grid = {
        'linear': {'C': [0.01, 0.1, 1, 10, 100]},
        'poly': {'C': [0.01, 0.1, 1, 10, 100], 'degree': [2, 3,
           4], 'gamma': ['scale', 'auto']},
        'rbf': {'C': [0.01, 0.1, 1, 10, 100], 'gamma': ['scale', '
           auto']}
    }
    model = SVC(kernel=kernel_type, probability=True)
    grid_search = GridSearchCV(model, param_grid[kernel_type], cv
       =5, scoring='accuracy', n_jobs=-1)

    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_
    extract_and_visualize_features(X_train, y_train, f"
       Hyperparameter-Tuned SVM ({kernel_type})")

    return best_model, grid_search.best_params_
```

3.3.5 Default SVM Training

Default SVM models were initially trained using standard, out-of-the-box parameters for each kernel type, establishing a baseline for subsequent tuning efforts.

Linear Kernel

A linear SVM was trained with a default regularization parameter $C = 1.0$. The linear kernel is particularly effective for datasets that exhibit linear separability. The C parameter controls the trade-off between achieving a smooth decision boundary and correctly classifying training points. A lower C encourages a wider margin and simpler decision boundary, while a higher C aims to minimize classification errors on the training data, potentially leading to overfitting.

Polynomial Kernel

A polynomial SVM was trained with default parameters, including a degree of 3 and gamma set to 'scale'. The polynomial kernel introduces non-linearity by mapping the input data to a higher-dimensional space using polynomial functions. The degree parameter determines the highest power of the polynomial, influencing the complexity of the learned decision boundary. Gamma, when set to 'scale', adapts to the variance of the input data, providing a more robust model.

RBF Kernel

An RBF (Radial Basis Function) SVM was trained with default parameters, including gamma set to 'scale'. The RBF kernel is exceptionally effective for modeling complex, non-linear decision boundaries by mapping the input data into an infinite-dimensional space. This allows the model to capture intricate patterns in the data.

```
def train_svm_default(X_train, y_train, kernel_type):
    param_dict = {
        'linear': {'C': 1.0},
        'poly': {'C': 1.0, 'degree': 3, 'gamma': 'scale'},
        'rbf': {'C': 1.0, 'gamma': 'scale'}
    }

    params = param_dict[kernel_type]
    model = SVC(kernel=kernel_type, probability=True, **params)
    model.fit(X_train, y_train)

    extract_and_visualize_features(X_train, y_train, f"Default SVM
        ({kernel_type})")

    return model, params
```

3.3.6 Random Forest

Random Forest is ensemble learning method that constructs multiple decision trees and aggregates their predictions, resulting in improved accuracy and robustness compared to individual decision trees. This approach mitigates overfitting and enhances generalization by reducing variance.

Understanding Random Forest Parameters

The performance of a Random Forest model is governed by several key hyperparameters that control the structure and learning process of the individual decision trees and the ensemble as a whole.

1. n_estimators ($N_{\mathbf{trees}}$): This parameter defines the number of decision trees in the forest. A larger number of trees generally leads to more stable and accurate predictions, as the ensemble's decision is based on the aggregation of more individual tree outputs. However, increasing the number of trees also increases the computational cost of training and prediction. There is no direct formula for N_{trees} itself, but it determines the size of the ensemble.

2. max_depth ($d_{\mathbf{max}}$): The 'max_depth' parameter limits the maximum depth of each individual decision tree. A deeper tree can capture more complex patterns in the training data but is also more prone to overfitting. Limiting the depth helps to regularize the trees and improve generalization to unseen data.

3. min_samples_split ($n_{\mathbf{split}}$): This parameter specifies the minimum number of samples required for a node to be split into further sub-nodes. A higher value restricts the splitting of nodes that have few samples, which can help prevent the creation of overly specific trees that overfit to small variations in the training data.

4. min_samples_leaf ($n_{\mathbf{leaf}}$): The 'min_samples_leaf' parameter sets the minimum number of samples required to be present at a leaf node. Similar to 'min_samples_split', this parameter helps to regularize the trees by ensuring that leaf nodes have a certain level of support from the training data, preventing the creation of leaves based on very few instances.

5. max_features ($F_{\mathbf{max}}$): This parameter controls the number of features to consider when looking for the best split at each node in each tree. By randomly selecting a subset of features at each split, Random Forest introduces diversity among the trees, which is crucial for reducing the correlation between their predictions and improving the ensemble's overall performance.

Common options include considering all features, the square root of the total number of features ($\sqrt{n_{\text{features}}}$), or a fixed percentage.

3.3.7 Default Random Forest Training

A default Random Forest model was trained with standard parameters, including 100 trees, no maximum depth limit, and other default settings. This setup provides a baseline for evaluating the impact of hyperparameter tuning. Below is the code snippet for training the random forest classifier with default parameters as follow:

```
1
2  def train_default_random_forest(X_train, y_train):
3      model = RandomForestClassifier(
4          n_estimators=100,
5          max_depth=None,
6          min_samples_split=2,
7          min_samples_leaf=1,
8          max_features='sqrt',
9          bootstrap=True,
10         random_state=42
11     )
12
13     model.fit(X_train, y_train)
14     extract_and_visualize_features(X_train, y_train, "Default
       Random Forest")
15
16     return model, model.get_params()
```

3.3.8 Hyperparameter Tuned Random Forest Training GridSearchCV

To fine-tune the Random Forest model and optimize its performance, GridSearchCV was employed. This method systematically explores a predefined grid of hyperparameter values, using cross-validation to assess each combination. The hyperparameters tuned included:

- **n_estimators:** The number of trees in the forest.
- **max_depth:** The maximum depth of the trees.

- **min_samples_split:** The minimum number of samples required to split an internal node.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node.
- **max_features:** The number of features to consider when looking for the best split.

The grid search included various values for these parameters to find the optimal combination. This comprehensive tuning process ensures that the Random Forest model achieves the best possible performance on the cataract detection task, balancing bias and variance effectively.Below is the code snippet for training the random forest classifier with tuned hyper parameters as follow:

```
def train_tuned_random_forest(X_train, y_train):
    param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [10, 20, 30, None],
        'min_samples_split': [2, 5, 10, 15],
        'min_samples_leaf': [1, 2, 4, 5],
        'max_features': ['sqrt', 'log2'],
        'bootstrap': [True]
    }

    model = RandomForestClassifier(random_state=42)
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='
       accuracy', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    extract_and_visualize_features(X_train, y_train, "Tuned Random
        Forest")

    return best_model, grid_search.best_params_
```

Chapter 4

System Design and Development

This chapter details the actual implementation of cataract detection system concentrating on system architecture, diagrams and mobile application development process. It highlight the integration of trained machine learning model with in a user centric mobile application for efficient detecting process.

4.1 Requirement Gathering and Analysis

The initial step of system development lifecycle involved understanding of the project scope and objectives through requirement gathering and analysis. This process aimed to identify all stakeholders define functionalities and develop the non functional requirements that will manage application performance.

4.1.1 Identifying Stakeholders

Meeting the requirements and expectations of variosu stakeholders is important for our application. Primarily the end users are individuals seeking convenient assessment of cataract. For these users the application must be intuitive providing clear instruction for image capture and selection and presenting the results in an easily understandable way, therefore empowering them with initial insight into the eye health without the immediate need for clinical visit. Healthcare professionals particularly ophthalmologists form another improtant stakeholders. However the application is not intended to replace professional diagnosis but it can serve as tool for detection of cataract. This could potentially optimize the workflow of healthcare workers by focusing their expertise on individuals identified as potentially having cataract. Lastly the researchers or developers may by involved in this project as stakeholders becuase they invested in advancing the application of machine learning in healthcare sector. Their interest lies in the effective im-

plementation of AI algorithms for medical image analysis and creation of robust and scalable mobile health solution.

4.1.2 Defining Functional Requirements

Functional requirements are the specific actions and features that the system must perform. These are detailed as follows:

- **User Authentication:** The system necessitates secure registration and login functionalities for every user. The Flutter code in `LoginScreen.dart` and `RegistrationScreen.dart` provides the user interface while Firebase Authentication handles the underlying secure user management.
- **Image Acquisition:** Users must be able to capture eye image using their mobile device camera or select existing images from the gallery. The `CataractPredictionController.dart` in Flutter utilizing the `image_picker` package which enables this image input.
- **Image Preprocessing:** Collected image undergo preprocessing to standardize it for machine learning model. Based on the research methodology, this includes steps like resizing to 100x100 pixels, normalization of pixel values, and application of unsharp masking technique. The Flask API's `preprocess_image` and `unsharp_mask` functions in `app.py` implement these steps.
- **Cataract Detection:** The system utilizes trained machine learning model, specifically SVM with Linear, Polynomial, RBF kernels and Random Forest classifier to analyze the preprocessed images for signs of cataracts. The training of these models is detailed in `main.py`, `trainsvm.py` and `train_rf.py` and Flask API's '/predict' route in `app.py` loads these models (via `joblib`) for image classification.
- **Results Presentation:** The result of cataract detection is presented to the user. The `PredictionController.dart` in Flutter handles the `detectionResult` and `averageAccuracy` observables which are displayed in the application UI to indicate the predicted presence or absence of cataract and a confidence level.
- **Data Persistence :** The system has the ability to securely store user data and detection history. The architecture indicate the use of Firebase Authentication and Firestore for this and the `CataractPredictionController.dart` includes the `saveDetectionResults` function for storing detection information.
- **Sign Out:** Users must be able to securely log out of their accounts functionality implied by the user interface design in `LoginScreen.dart`.

4.1.3 Defining Non-Functional Requirements

Non functional requirements define the quality attributes of the system, ensuring it meets certain performance, security, and usability standards.

- **Performance:** The system must give detection results with minimum latency. The Flask API need to efficiently process images and return predictions while the mobile application should remain responsive during image selection and result display to ensure user engagement.
- **Security:** User data especially during login credentials managed by Firebase Authentication and any potentially stored detection history must be non negotiable. Secure handling of image data transmitted to the backend API is also essential to protect user privacy.

4.1.4 Use Case Diagram

The use case diagram shown below will provides the visual overview of the application functionalities and their relationship with end user. It depicts the actor (End User) and the use cases they interact with.

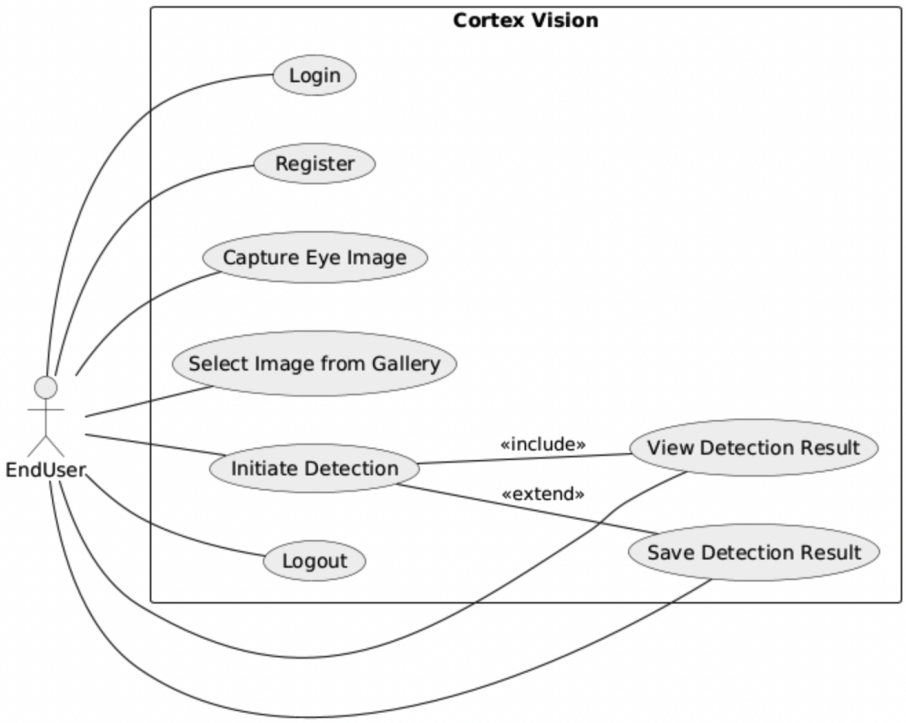

Figure 4.1: Use Case Diagram

4.2 System Design

The application is designed as a multi layerd architecture prioritizing modularity scalability, and maintainability. It comprises the following key components:

4.2.1 Architecture Design

- **Presentation Layer:** The Flutter based mobile application which is responsible for the user interface and user interaction. This layer provides the user friendly interface through which users interact with the system.
- **Application Layer:** The Flask API handling the core logic of the application. This includes image processing, interaction with the machine learning model for cataract detection and management of data flow between the presentation and data layer.
- **Data Layer:** Firebase services consisting of Authentication, Firestore and Storage. This layer is responsible for managing user data, storing detection results and handling the storage and retrieval of uploaded images.

This layered approach effectively separates concerns promoting more organized and structured design. This separation simplifies the development process and makes testing more manageable and enhances the overall maintainability of the system.

Component Diagrams

The component diagram will give a high level overview of system architectural blocks and their dependents relationships and illustrate the organization and dependencies among the different components, interfaces and data stores within the System. The priamary components include User Interface (Mobile Application) , API Layer (Flask API), an Authentication Service, Prediction Service, Data Storage (Firebase) and Machine Learning Model. The dependencies amoung these component highlight the flow of data and control within the application which illustrating how user requests are processes and predictions are generated and how data is then managed. This structure is illustrated in the diagram below.

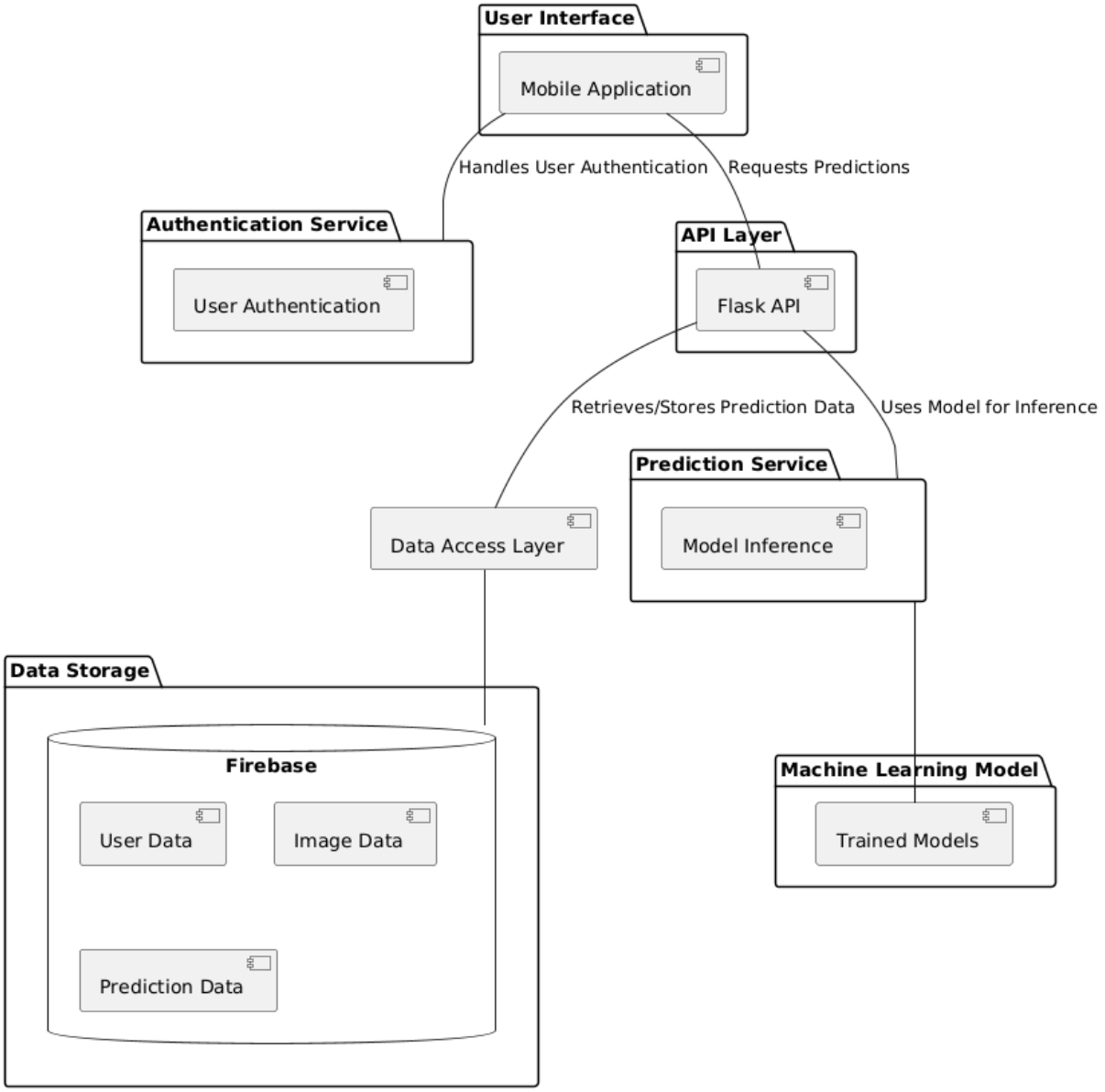

Figure 4.2: Component Diagram

4.2.2 Technology Stack Selection

The following technologies are chosen for the development of the mobile application system:

- **Frontend:** Flutter an open source User Interface (UI) software development kit (SDK) conceived and maintained by Google represents a paradigm shift in cross platform application development. It empowers developer to construct high performance visually rich applications from a singular codebase streamlining the development process across di-

verse platforms including mobile operating systems (iOS and Android) web browsers and desktop environments (Windows, macOS, and Linux). This unified approach not only accelerates development cycles but also ensures a consistent user experience across all supported platforms.

- **Backend:** Flask, a lightweight and flexible Python web framework, was chosen for developing the API. Flask's simplicity and extensibility make it well-suited for handling the application's backend logic.
- **Image Processing:** OpenCV was selected for due to its powerful image processing capabilities. It is used for preprocessing the images before they are fed into the machine learning models.
- **Machine Learning:** The scikit learn library in Python was used to implement the Support Vector Machine (SVM) and Random Forest machine learning model. Scikit learn provides efficient tools for machine learning including classification, regression and clustering.
- **Database and Authentication:** Firebase a comprehensive platform provided by Google, was chosen for:
  - **Authentication:** Firebase Authentication provides secure and easy to use authentication services for managing user logins and registration.
  - **Firestore:** Firebase Firestore a NoSQL cloud database was selected for storing user data and detection results.
  - **Storage:** Firebase Storage was used for storing the images uploaded by users.
- **Operating System:** The mobile application is designed to be compatible with both Android and iOS operating systems ensuring broad accessibility for users.

These technologies were selected based on careful evaluation of factors such as performance, cross platform compatibility, scalability to handle a growing user base and ease of development allowing for efficient and effective implementation of the system.

4.2.3 Key Benefits of Using Flutter

- **Hot Reload for Accelerated Development Cycles:** Flutter hot reload feature allows us to instantly view the effects of code change without restarting the application. This capability significantly reduces development time facilitating rapid prototyping and iterative design.
- **Extensive Library of Pre designed Widgets for Custom UI Design:** Flutter provides a comprehensive suite of pre designed widgets that adhere to Material Design (Google) and Cupertino (Apple) guidelines enabling us to create visually stunning and platform adaptive user interface. These widgets are highly customizable, offering developers granular control over the application's appearance and behavior.
- **Native Performance for Seamless User Experiences:** Flutter applications are compiled directly into native machine code resulting in performance comparable to applications developed using platform specific languages (e.g. Swift for iOS, Kotlin or Java for Android). This native compilation ensures smooth animations fluid transitions and responsive interactions, delivering a superior user experience.

The adoption of Flutter for this project underscores its capacity to deliver a robust performant and visually engaging cataract detection application. The framework ability to streamline the development coupled with its native performance and extensive widget library makes it ideal choice for building complex, cross platform applications that demand high levels of user interaction and visual fidelity.

4.2.4 API Design

The API design for the system mainly focuses on providing simple and efficient way for the mobile application to request prediction from trained machine learning models. The core of the API revolves around single endpoint that manage image upload and returns prediction results as well as confidence score.

Endpoint Definitions

The primary API endpoint is defined as follows:

- **/predict**:
  - **Method:** `POST`
  - **Request Body:** Accepts multipart/form-data request with a file field named `'image'` containing the eye image.

- **Response Body:** Returns JSON object containing the prediction from various machine learning models including the predicted class (Cataract or Normal) and the confidence score/accuracy.

4.2.5 Request Handling

The `predict()` function in the Flask API handles incoming requests to the `/predict` endpoint:

1. **Model Loading Check:** It first checks if the machine learning models were loaded successfully during the API startup. If not, it returns a 503 Service Unavailable error.
2. **Image File Validation:** It verifies if an `image` file is present in the request's files. If not, it returns a 400 Bad Request error. It also checks if a file was actually selected.
3. **Temporary Image Saving:** The uploaded image file is temporarily saved on the server (`temp_image.jpg`). This allows OpenCV (`cv2`) to read the image from a file path.
4. **Image Preprocessing:** The `preprocess_image()` function is called to resize the image to a standard size (`IMG_SIZE`), normalize pixel values, and apply an unsharp masking filter to enhance details.
5. **Prediction Generation:** The preprocessed image is then passed to each of the loaded machine learning models (`default_svm_linear`, `default_svm_poly`, `default_svm_rbf`, `tuned_svm_linear`, `tuned_svm_poly`, `tuned_svm_rbf`, `default_rf_model`, `tuned_rf_model`). For each model, the API attempts to:
   - Make a prediction using the `.predict()` method.
   - Get the probability/confidence score using the `.predict_proba()` method (where available).
   - Format the prediction (Cataract or Normal) and the accuracy as a percentage.
   - Include error handling within `try-except` blocks to gracefully manage potential issues with individual model predictions.
6. **Temporary File Cleanup:** After processing, the temporarily saved image file (`temp_image.jpg`) is deleted to free up server storage.
7. **Error Handling:** A general `try-except` block wraps the entire prediction process to catch any unexpected errors and return a 500 Internal Server Error with an error message.

8. **Temporary File Cleanup (Finally):** A `finally` block ensures that the temporary image file is deleted even if an error occurs during the prediction process.

```
@app.route('/predict', methods=['POST'])
def predict():
if not models_loaded:
return jsonify({'error': 'Models not loaded'}), 503
image_file = request.files['image']
temp_path = 'temp_image.jpg'
try:
image_file.save(temp_path)
print(f"[{datetime.now()}] Received and saved image: {image_file.
   filename} to {temp_path}")
processed_image = preprocess_image(temp_path, IMG_SIZE)
os.remove(temp_path)
    if processed_image is None:
        return jsonify({'error': 'Failed'}), 500
    predictions = {}
    models = {
        'Default Linear SVM': def_svm_linear,
        'Default Polynomial SVM': def_svm_poly,
        'Default RBF SVM': def_svm_rbf,
        'Default Random Forest': def_rf_model,
        'Tuned Linear SVM': tun_svm_linear,
        'Tuned Polynomial SVM': tun_svm_poly,
        'Tuned RBF SVM': tun_svm_rbf,
        'Tuned Random Forest': tun_rf_model,
    }
    for model_name, model in models.items():
        try:
            pred = model.predict(processed_image)[0]
            prob = np.max(model.predict_proba(processed_image)) *
               100
            predictions[model_name] = {
                'detection': 'Cataract' if pred == 1 else 'Normal'
                   ,
                'accuracy': f'{prob:.2f}%' }
        except Exception as e:
```

```
            predictions[model_name] = {'error': f'Prediction
                failed: {e}'}
    return jsonify(predictions)
except Exception as e:
    error_message = f"{e}"
    return jsonify({'error': error_message}), 500
finally:
    if os.path.exists(temp_path):
        os.remove(temp_path)
```

The code first checks if the models are loaded. It then retrieves the uploaded image and saves it temporarily, preprocesses it and removes the temporary file. Each pretrained model generates a prediction and the results are stored in the predictions dictionary. This dictionary is then converted to JSON response using Flask jsonify function.

4.2.6 Response Generation

The predict endpoint returns a JSON response containing the predictions from all the loaded machine learning models. The structure of the JSON response is a dictionary where:

- **Keys:** Represent the names of the machine learning models (e.g. "Default Linear SVM", "Tuned RBF SVM", "Default Random Forest").
- **Values:** Are dictionaries containing the prediction results for that specific model. Each model's result dictionary typically includes:
  - `"detection"`: The predicted class ("Cataract" or "Normal").
  - `"accuracy"`: The confidence score of prediction formatted as a percentage string (e.g. 97.44%).
  - `"error"`: If prediction for a specific model fail the key will be present with an error message.

```
{
  "Default Linear SVM": {
    "detection": "Cataract",
    "accuracy": "83.86%"
  },
  "Default Polynomial SVM": {
    "detection": "Cataract",
    "accuracy": "84.24%"
  },
  "Default RBF SVM": {
    "detection": "Cataract",
    "accuracy": "97.19%"
  },
  "Default Random Forest": {
    "detection": "Cataract",
    "accuracy": "72.00%"
  },
  "Tuned Linear SVM": {
    "detection": "Cataract",
    "accuracy": "90.79%"
  },
  "Tuned Polynomial SVM": {
    "detection": "Cataract",
    "accuracy": "88.84%"
  },
  "Tuned RBF SVM": {
    "detection": "Cataract",
    "accuracy": "97.44%"
  },
  "Tuned Random Forest": {
    "detection": "Cataract",
    "accuracy": "83.70%"
  }
```


4.2.7 Sequence Diagram

The following subsections illustrate the workflow of the cataract detection system, detailing the interactions between the user, the mobile application, the backend services (Flask API and

Firebase), and the device components (camera/gallery).

Authentication and Image Selection

The below given sequence diagram illustrates the initial steps in the cataract detection and providing detail of the interactions involved in user authentication and capture of an image for analysis.

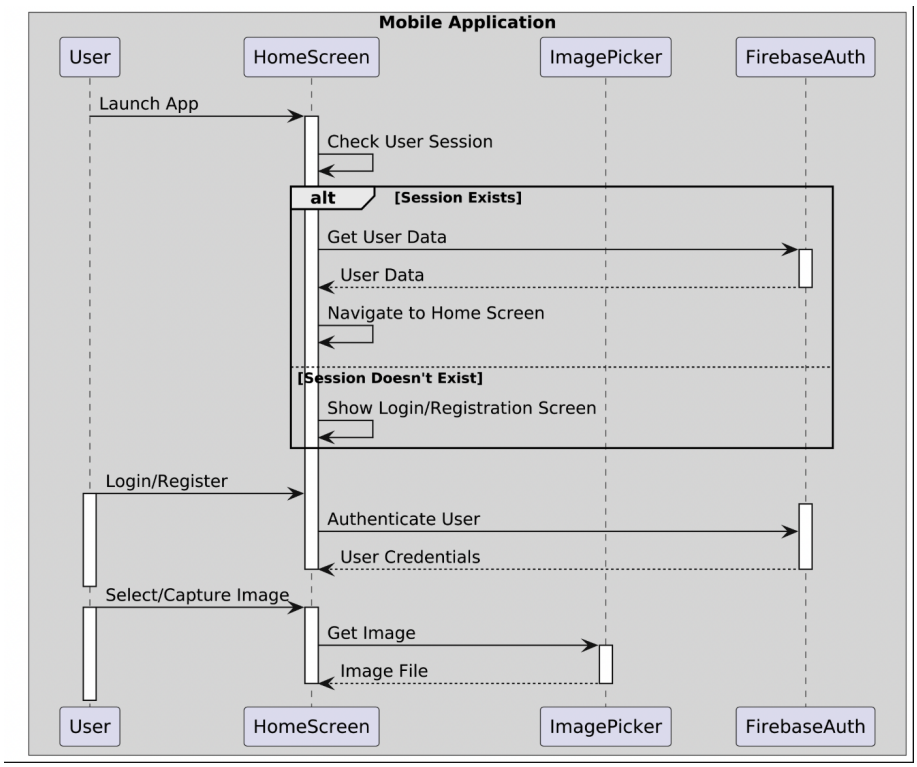

Figure 4.3: Authentication and Image Selection

Image Processing and Saving

The below given sequence diagram provide the details of the subsequent steps in the cataract detection illustrating the interactions involved in processing the selected image by the backend services and the user's option to save the analysis results.

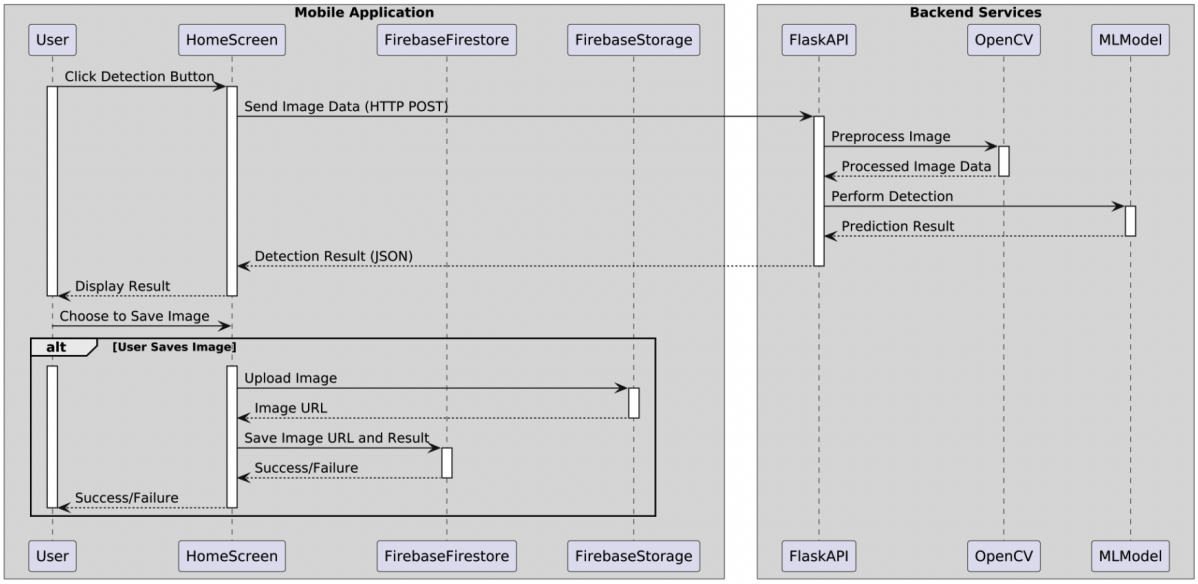

Figure 4.4: Image Processing and Saving

4.3 Implementation (Coding & Development)

The client side application was developed using the Flutter framework and Dart programming language. This section provides a detailed look at the architecture component structure and functional implementation of the frontend module. As the Key aspects of the Flutter implementation include:

- **State Management:** Flutter's `StatefulWidget` is used extensively to manage dynamic UI updates. This allows the application to respond to user interactions and changes in data, providing a dynamic and interactive user experience.
- **UI Components:** A rich set of predesigned widgets from the `material` and `cupertino` packages are used to create a platform-adaptive UI. This ensures that the application adheres to the design guidelines of both Android and iOS, providing a nativelike feel on each platform.
- **Image Handling:** The `image_picker` package is used to facilitate image selection from the device's gallery or capture from the camera. This package simplifies the process of accessing the device's image resources.

```
1  Rx<File?> imageFile = Rx<File?>(null);
2      Future<void> imageFromGallery() async {
3          final pickedFile = await ImagePicker().getImage(
               source: ImageSource.gallery, maxHeight: 200,
               maxWidth: 200);
4          if (pickedFile != null) {
5              imageFile.value = File(pickedFile.path);
6              detectionResult.value = '';
7              averageAccuracy.value = 0.0;
8              isDetectionComplete.value = false;
9           }
10     }
```

- **HTTP Communication:** The `http` package is used to send image data to the Flask API for processing and to receive detection results. This package enables the application to make asynchronous HTTP requests.

```
Future<void> startDetection() async {
  isDetecting.value = true;
  detectionResult.value = '';
  averageAccuracy.value = 0.0;

  final uri = Uri.parse('http://127.0.0.1:5000/predict');
  var request = http.MultipartRequest('POST', uri);
  request.files.add(await http.MultipartFile.fromPath('image'
     , imageFile.value!.path));
  try {
    var streamedResponse = await request.send();
    var response = await http.Response.fromStream(
       streamedResponse);
    detectionResponseData = jsonDecode(response.body);

    if (response.statusCode == 200) {
      double totalAccuracy = 0;
      int validAccuracyCount = 0;

      if (detectionResponseData != null) {
        detectionResponseData!.forEach((modelName, data) {
          if (data != null && data.containsKey('accuracy')) {
            try {
              double accuracy = double.parse(data['accuracy'
                 ].toString().replaceAll('%', ''));
              totalAccuracy += accuracy;
              validAccuracyCount++;
            } catch (e) {
              print('Error parsing accuracy for $modelName:
                 $e');
            }
          }
          if (modelName == 'Default Linear SVM' && data !=
             null && data.containsKey('detection')) {
```

```
            detectionResult.value = data['detection'];
          }
        });

        if (validAccuracyCount > 0) {
          averageAccuracy.value = totalAccuracy /
              validAccuracyCount;
        }
      } else {
        detectionResult.value = 'Error: Invalid response
            format';
      } } else {
      detectionResult.value = 'Error: ${detectionResponseData
          ?['error'] ';
    }
  } catch (e) {
    detectionResult.value = 'Error: $e';
  } finally {
    isDetecting.value = false;
  }
}
```

- **Firebase Integration:**

  The application integrates with several Firebase services: firebase Auth is used for user authentication, providing secure user login and registration functionality.

Code Snippet - Login

```
Future<void> login(String email, String password) async {
    if (validateEmail(email) != null || validatePassword(
        password) != null) {
        return;
    } try {
        isLoading.value = true;
        await _auth.signInWithEmailAndPassword(email: email,
            password: password);
        Get.offAll(() => CataractPredictionScreen());
    } catch (e) {
```

```
        isLoading.value = false;
        String errorMessage = "Login failed. Please check
            your credentials.";
    } finally {
        isLoading.value = false;
    }
}
```

```
void register() async {
    if (formKey.currentState!.validate()) {
        formKey.currentState!.save();
        isLoading.value = true;
        try {
            final newUser = await FirebaseAuth.instance.
                createUserWithEmailAndPassword(
                 email: emailController.text,
                 password: passwordController.text,
            );
            await FirebaseFirestore.instance.collection('
                users').doc(newUser.user!.uid).set({
                 'username': usernameController.text,
                 'email': emailController.text,
                 'contactNumber': contactNumberController.text
                 'gender': selectedGender.value,
            });
            Get.offAll(LoginScreen());
        } finally {
            isLoading.value = false;
        }
    }
}
```

4.3. Implementation (Coding & Development)

Cloud Firestore is used for storing user data and detection results, allowing for persistent data storage.

```
Future<void> saveDetectionResults(String userId, String
   detectionResult, String accuracy) async {
    try {
        await FirebaseFirestore.instance.collection('
           detectionResults').add({
            'userId': userId,
            'imageURL': downloadUrl,
            'detectionResult': detectionResult,
            'averageAccuracyResult': accuracy,
            'uploadDate': formattedDate,
            'uploadTime': formattedTime,
        }););
    } catch (e) {
        print('Error saving detection results: $e');
    }
}
```

Firebase Storage is used for storing uploaded images, providing a scalable solution for image storage.

```
Future<String> uploadImageToStorage(File imageFile, String
   userId) async {
    String imageUuid = Uuid().v4();
        String fileName = 'detection_image_$imageUuid.jpg';
        firebase_storage.Reference ref = firebase_storage.
           FirebaseStorage.instance.ref().child('eyeImages').
           child(userId).child(fileName);
        try {
            await ref.putFile(imageFile);
            downloadUrl = await ref.getDownloadURL();
            return downloadUrl;
        } catch (e) {
            return '';
    }
}
```

4.4 Application User Interface

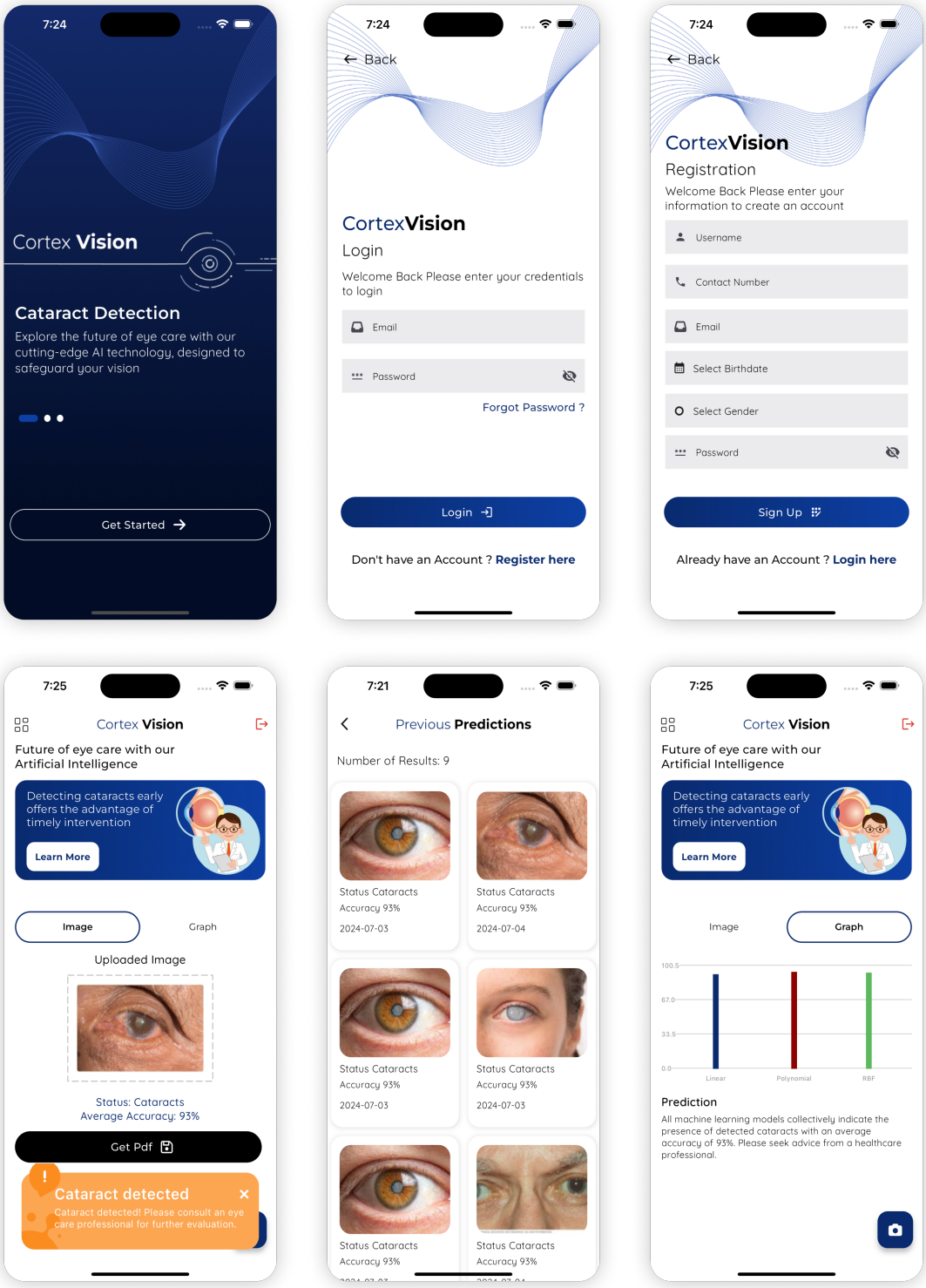

Figure 4.5: Mobile Application

Chapter 5

Evaluation

This chapter details the evaluation of the cataract detection models developed using Support Vector Machines (SVM) with linear, polynomial, and Radial Basis Function (RBF) kernels, as well as Random Forest. We assess the performance of both default and hyperparameter-tuned versions of these models on a held-out test dataset.

5.1 Evaluation Criteria

To comprehensively evaluate the performance of our models we have used multiple standard classification metrics. These metrics will provide insights into different aspects of the model ability to correctly classify images as 'Normal' or 'Cataract'.

- **Precision (Positive Predictive Value):** Defined as $\frac{TP}{TP+FP}$, where TP is the number of true positives and FP is the number of false positives. Precision determine the model's accuracy in predicting positive cases minimizing false alarms. The high precision presents that when the model predicts a positive outcome it is likely to be correct.
- **Recall (Sensitivity or True Positive Rate):** Defined as $\frac{TP}{TP+FN}$, where FN is the number of false negatives. Recall measures the model's ability to capture all actual positive cases minimizing missed detections. A high recall indicates that the model is effective at identifying all positive instances.
- **F1-Score (Harmonic Mean of Precision and Recall):** Defined as $2 \times \frac{Precision \times Recall}{Precision + Recall}$. The F1 score provides a balanced measure of precision and recall particularly useful in imbalanced datasets where one class significantly outnumbers the other. It is vital for scenarios where both false positives and false negatives have significant implications.

– **Accuracy:** Defined as $\frac{TP+TN}{TP+TN+FP+FN}$, where TN is the number of true negatives. Accuracy represents the overall correctness of the model predictions indicating the proportion of correctly classified instances out of the total.

The evaluation is conducted on the testing dataset with dimensions (126, 30000) comprising 126 samples each with 30000 features. This high dimensionality underscores the complexity of image data and challenge of accurate classification. The models evaluated are Support Vector Machines (SVM) with linear, polynomial and Radial Basis Function (RBF) kernels and Random Forest classifier. Both default and tuned kind of these models are assessed to elucidate the impact of hyperparameter optimization on model performance.

– **Precision (Positive Predictive Value):** Defined as $\frac{TP}{TP+FP}$, where TP is the number of true positives and FP is the number of false positives. Precision determine the model's accuracy in predicting positive cases minimizing false alarms. The high precision presents that when the model predicts a positive outcome it is likely to be correct.
– **Recall (Sensitivity or True Positive Rate):** Defined as $\frac{TP}{TP+FN}$, where FN is the number of false negatives. Recall measures the model's ability to capture all actual positive cases minimizing missed detections. A high recall indicates that the model is effective at identifying all positive instances.
– **F1-Score (Harmonic Mean of Precision and Recall):** Defined as $2 \times \frac{Precision \times Recall}{Precision+Recall}$. The F1 score provides a balanced measure of precision and recall particularly useful in imbalanced datasets where one class significantly outnumbers the other. It is vital for scenarios where both false positives and false negatives have significant implications.
– **Accuracy:** Defined as $\frac{TP+TN}{TP+TN+FP+FN}$, where TN is the number of true negatives. Accuracy represents the overall correctness of the model predictions indicating the proportion of correctly classified instances out of the total.

The evaluation is conducted on the testing dataset with dimensions (126, 30000) comprising 126 samples each with 30000 features. This high dimensionality underscores the complexity of image data and challenge of accurate classification. The models evaluated are Support Vector Machines (SVM) with linear, polynomial and Radial Basis Function (RBF) kernels and Random Forest classifier. Both default and tuned kind of these models are assessed to elucidate the impact of hyperparameter optimization on model performance.

5.1.1 Experimental Setup

This section give outline of the experimental setup including the dataset, training and testing procedures, hyperparameter tuning and computational resources used.

Dataset Description

The dataset consists of images categorized into two distinct classes: 'Normal' and 'Cataract'. The training set located in the train folder, contains 532 images while the testing set located in the test folder contains 126 images. Each image is resized to 100×100 pixels and normalized by dividing pixel values by 255. Furthure unsharp mask filter is applied as preprocessing step to enhance image details. The preprocessing.py script handles the loading and preprocessing of these images.

Training and Testing Procedures

The `main.py` script leads the training and evaluation process. It loads the training and testing datasets using the `load_images_from_folder` function from `preprocessing.py`. The image data is flattened into a 1 dimensional array which is to be compatible with the SVM and Random Forest models. The training includes fitting both default and hyperparameter tuned versions of the SVM (with linear, polynomial and RBF kernels) and Random Forest models using the training data (X_train, y_train). Hyperparameter tuning for SVM is performed using `GridSearchCV` with 5 fold cross validation, optimizing for accuracy. The tuned Random Forest model also utilizes `GridSearchCV` to find the best combination of hyperparameters.

After training the models were evaluated on the held out test set (X_test, y_test) using the `evaluate_model` function from `evaluation.py`. This function calculates accuracy and generates classification report containing precision, recall, and F1-score for each class.

Hyperparameter Tuning

The hyperparameter tuning for both the default and tuned models are defined in the respective training scripts (`train_svm.py` and `train_rf.py`).

Default SVM Parameters:

- Linear Kernel: $C = 1.0$
- Polynomial Kernel: $C = 1.0$, $degree = 3$, $\gamma = \text{scale}$
- RBF Kernel: $C = 1.0$, $\gamma = \text{scale}$

Tuned SVM Parameters (Obtained via GridSearchCV):

- Linear Kernel: $C = 0.01$
- Polynomial Kernel: $C = 1$, $degree = 3$, $\gamma =$ scale
- RBF Kernel: $C = 10$, $\gamma =$ scale

Default Random Forest Parameters:

- $n\_estimators = 100$
- $max\_depth =$ None
- $min\_samples\_split = 2$
- $min\_samples\_leaf = 1$
- $max\_features =$ 'sqrt'
- $bootstrap =$ True
- $random\_state = 42$

Tuned Random Forest Parameters (Obtained via GridSearchCV):

- $n\_estimators = 200$
- $max\_depth = 20$
- $max\_features =$ 'sqrt'
- $min\_samples\_leaf = 2$
- $min\_samples\_split = 10$
- $bootstrap =$ True

The `train_svm.py` and `train_rf.py` script contain the logic for training both default and tuned models.

Computational Resources

The experiments was conducted on Apple MacBook Pro (13-inch M1) with the following specifications:

- Apple M1 chip with 8 core CPU (4 performance cores and 4 efficiency cores) and 8 core GPU
- 16 core Neural Engine

- 8GB unified memory

The training times varied depending on complexity of model and whether hyperparameter tuning was involved. The `GridSearchCV` function used for hyperparameter optimization utilized the parallel processing capabilities of the M1 chip to increase the search process by setting the `n_jobs` parameter to -1.

5.2 Detailed Evaluation Results

This section will present the detailed evaluation result for each model on both the training and testing datasets.

5.2.1 Support Vector Machines (SVM)

We evaluated SVM models with three distinct kernels: linear, polynomial and RBF. For each kernel we trained both default version and hyperparameter tuned version.

Results on Training Set

The training accuracies for the SVM models are as follows:

- Default Linear SVM: 100.00%
- Default Polynomial SVM: 99.81%
- Default RBF SVM: 98.87%
- Tuned Linear SVM: 99.81%
- Tuned Polynomial SVM: 99.81%
- Tuned RBF SVM: 99.81%

The high training accuracies indicate that the SVM models can effectively learn the training data. The default linear SVM achieved perfect accuracy on the training set which might suggest overfitting.

Results on Testing Set

The classification reports for the default SVM models on the testing set are as fellows:

Default Linear SVM:

	precision	recall	f1-score	support
0	0.84	0.91	0.87	57
1	0.92	0.86	0.89	69
accuracy			0.88	126
macro avg	0.88	0.88	0.88	126
weighted avg	0.88	0.88	0.88	126

The testing accuracy for the default linear SVM is 88.10%.

Default Polynomial SVM:

	precision	recall	f1-score	support
0	0.90	0.95	0.92	57
1	0.95	0.91	0.93	69
accuracy			0.93	126
macro avg	0.93	0.93	0.93	126
weighted avg	0.93	0.93	0.93	126

The testing accuracy for the default polynomial SVM is 92.86%.

Default RBF SVM:

	precision	recall	f1-score	support
0	0.93	0.91	0.92	57
1	0.93	0.94	0.94	69
accuracy			0.93	126
macro avg	0.93	0.93	0.93	126
weighted avg	0.93	0.93	0.93	126

The testing accuracy for the default RBF SVM is 92.86%.

The classification reports for the hyperparameter tuned SVM models on the testing set are:

Tuned Linear SVM:

	precision	recall	f1-score	support
0	0.84	0.93	0.88	57
1	0.94	0.86	0.89	69
accuracy			0.89	126
macro avg	0.89	0.89	0.89	126
weighted avg	0.89	0.89	0.89	126

The testing accuracy for the tuned linear SVM is 88.89%.

Tuned Polynomial SVM:

	precision	recall	f1-score	support
0	0.90	0.95	0.92	57
1	0.95	0.91	0.93	69
accuracy			0.93	126
macro avg	0.93	0.93	0.93	126
weighted avg	0.93	0.93	0.93	126

The testing accuracy for the tuned polynomial SVM is 92.86%.

Tuned RBF SVM:

	precision	recall	f1-score	support
0	0.95	0.95	0.95	57
1	0.96	0.96	0.96	69
accuracy			0.95	126
macro avg	0.95	0.95	0.95	126
weighted avg	0.95	0.95	0.95	126

The testing accuracy for the tuned RBF SVM is 95.24%.

Comparison with Different Kernels

On the testing set, the polynomial and RBF kernels outperformed the linear kernel in both default and tuned configurations. The tuned RBF SVM achieved the highest accuracy among all SVM models at 95.24%.

Impact of Hyperparameter Tuning

Hyperparameter tuning slightly improved the performance of the linear and RBF SVM models on the testing set. The tuned linear SVM showed a slight increase in accuracy from 88.10% to 88.89% and the tuned RBF SVM showed more significant improvement from 92.86% to 95.24%. The polynomial SVM performance remained the same after tuning.

5.2.2 Random Forest

We trained and evaluated both default and hyperparameter tuned Random Forest model.

Results on Training Set

The training accuracies for the Random Forest models are:

- Default Random Forest: 100.00%
- Tuned Random Forest: 100.00%

Both the default and tuned Random Forest models achieved perfect accuracy on the training set indicating a ability to fit the training data.

Results on Testing Set

The classification reports for the Random Forest models on the testing set are:

Default Random Forest:

	precision	recall	f1-score	support
0	0.92	0.96	0.94	57
1	0.97	0.93	0.95	69

accuracy			0.94	126
macro avg	0.94	0.95	0.94	126
weighted avg	0.95	0.94	0.94	126

The testing accuracy for the default Random Forest is 94.44%.

Tuned Random Forest:

	precision	recall	f1-score	support
0	0.92	0.96	0.94	57
1	0.97	0.93	0.95	69
accuracy			0.94	126
macro avg	0.94	0.95	0.94	126
weighted avg	0.95	0.94	0.94	126

The testing accuracy for the tuned Random Forest is also 94.44%.

Impact of Number of Trees

The default Random Forest uses 100 trees ($n\_estimators = 100$) while the tuned version uses 200 trees ($n\_estimators = 200$). In this experiment increasing the number of trees did not lead to notice-able improvement in testing accuracy. Itis possible that the performance plateaued with 100 trees for this specific dataset and feature representation.

Impact of Splitting Criteria

The default Random Forest utilize the Gini impurity as the criterion for splitting nodes, which is standard setting in scikit-learn. The hyperparameter tuning process did not explicitly alter this criterion; the best parameters found focused on other aspects like the depth of trees and the minimum number of samples required for splitting and at the leaf nodes. Therefore we cannot directly assess the impact of different splitting criteria based on these results.

Impact of Hyperparameter Tuning

Apart from achieving perfect accuracy on training set hyperparameter tuning did not improve the testing accuracy of the Random Forest model in this experiment. Both the default and the tuned Random Forest achieved testing accuracy of 94.44%. This suggests

that default hyperparameters were already quite effective for this dataset and the extracted features furthermore improvements might necessitate a more extensive exploration of the hyperparameter space.

5.2.3 Comparison of Different Models

The following table give us the summary of the training and testing accuracies of all the evaluated models providing a comprehensive comparison of performance on both seen and unseen data:

Model	Training Accuracy (%)	Testing Accuracy (%)
Default Linear SVM	100.00	88.10
Default Polynomial SVM	99.81	92.86
Default RBF SVM	98.87	92.86
Tuned Linear SVM	99.81	88.89
Tuned Polynomial SVM	99.81	92.86
Tuned RBF SVM	99.81	**95.24**
Default Random Forest	100.00	94.44
Tuned Random Forest	100.00	94.44

As clear from the aboev table the **Tuned RBF SVM** achieved the highest testing accuracy of 95.24% among all the models which are evaluated. The Random Forest models (both default and tuned) also shows strong performance on testing set with an accuracy of 94.44%. The polynomial and default RBF SVMs showed comparable testing performance at 92.86% while the linear SVM models (both default and tuned) had the lowest testing accuracy.

There several models achieved very high or a perfect training accuracy (e.g. Default Linear SVM and both Random Forest models) which when compared to their testing accuracies, might indicate a degree of overfitting. The Tuned RBF SVM while having slightly lower training accuracy than some others which exhibited the best generalization to the unseen test data. This comparison underscores the importance of evaluating models on a separate test set to assess their real world performance and the potential impact of overfitting.

Error Analysis

While the quantitative metrics give overall assessment of the models performance and a deeper understanding that can be gained by analyzing the types of errors made. Error analysis typically involves examining the instances that were misclassified by the best-performing models to identify patterns that might contribute to these errors.

- **Image Quality Issues:** Variations in image quality (e.g., blurriness, poor lighting, artifacts) within the test set could lead to misclassifications, especially if the training set did not adequately represent these variations. The unsharp mask preprocessing step aims to mitigate some blurriness, but severe quality issues might still pose challenges.
- **Variations in Normal Eyes:** Normal eyes can also exhibit variations in appearance (e.g., pupil size, iris patterns, reflections). If these variations are not well represented in the training data, then this could also lead to potentially being misclassified as cataract (false positives).
- **Feature Limitations:** The extracted features (color means, edge counts, Hu moments) might not capture all the subtle but important visual cues necessary for perfect classification.
- **Model Limitations:** While RBF SVM and Random Forest are powerful models, they still have limitations in capturing complex relationships within the data. Certain complex patterns indicative of cataract might not be fully learned by these models.

5.2.4 Qualitative Evaluation: Classification Report Visualization

To provide a more intuitive understanding of the performance of our cataract detection models, we present the confusion matrices for each model on the test dataset.These reports provide a detailed breakdown of precision, recall, and F1-score for both the 'Normal' and 'Cataract' classes, offering a comprehensive view of how well each model classifies.

5.2. Detailed Evaluation Results

Classification Report - Default Linear SVM:

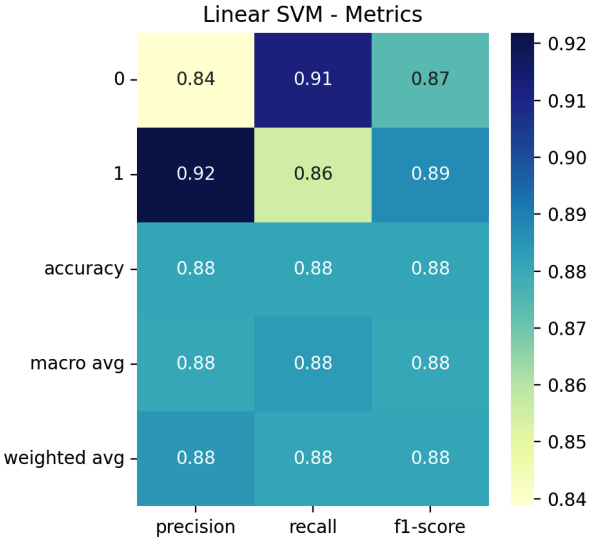

Figure 5.1: Classification Report for Default Linear SVM

Classification Report - Default Polynomial SVM:

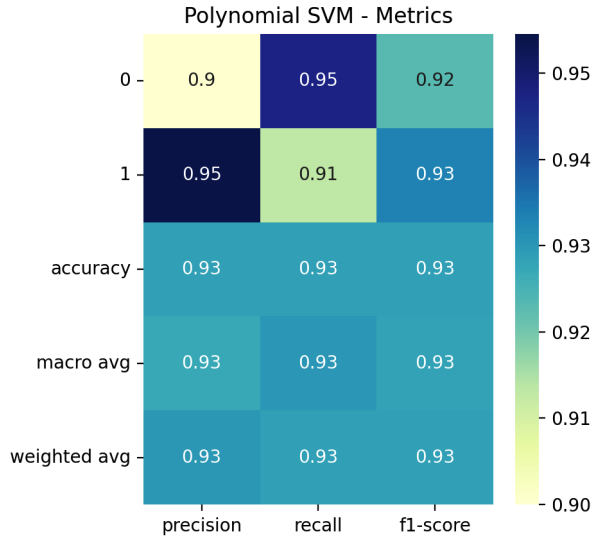

Figure 5.2: Classification Report for Default Polynomial SVM

Classification Report - Default RBF SVM:

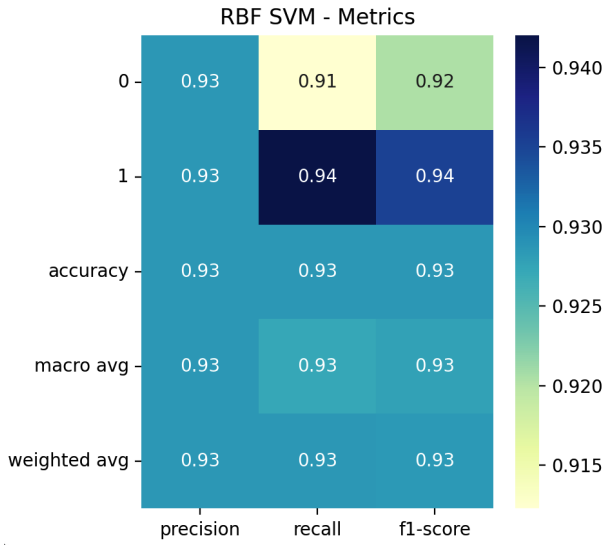

Figure 5.3: Classification Report for Default RBF SVM

Classification Report - Tuned Linear SVM:

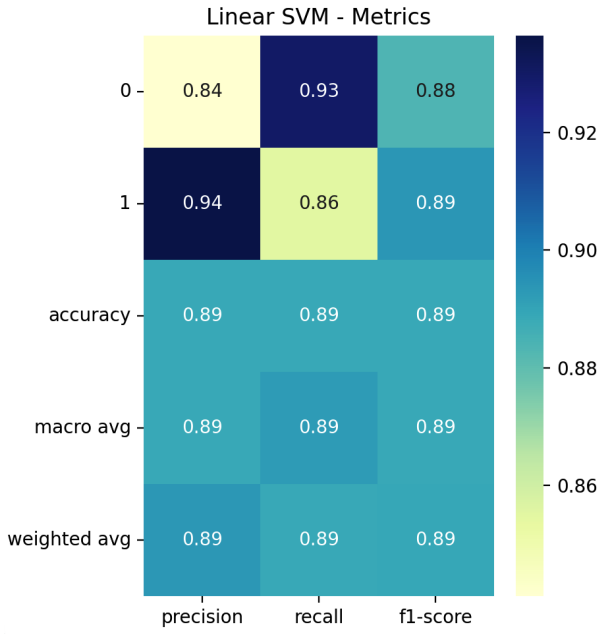

Figure 5.4: Classification Report for Tuned Linear SVM

Classification Report - Tuned Polynomial SVM:

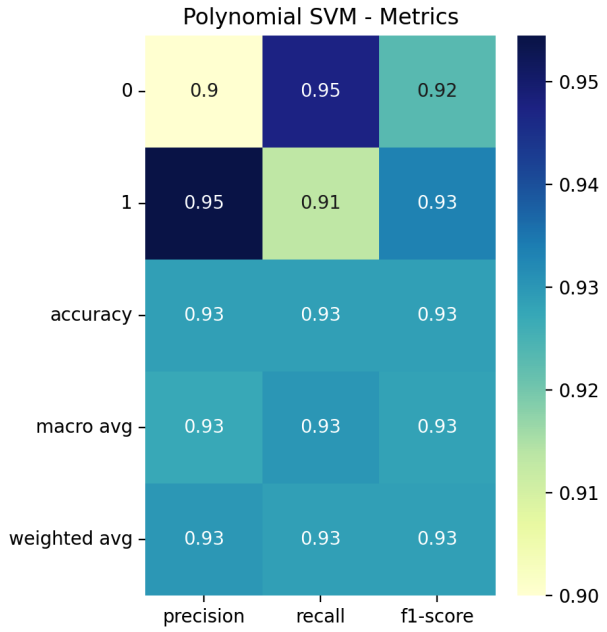

Figure 5.5: Classification Report for Tuned Polynomial SVM

Classification Report - Tuned RBF SVM:

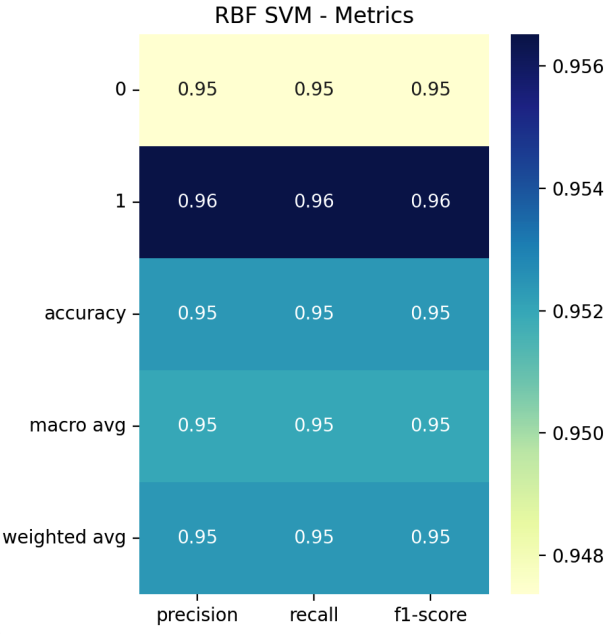

Figure 5.6: Classification Report for Tuned RBF SVM

Classification Report - Default Random Forest:

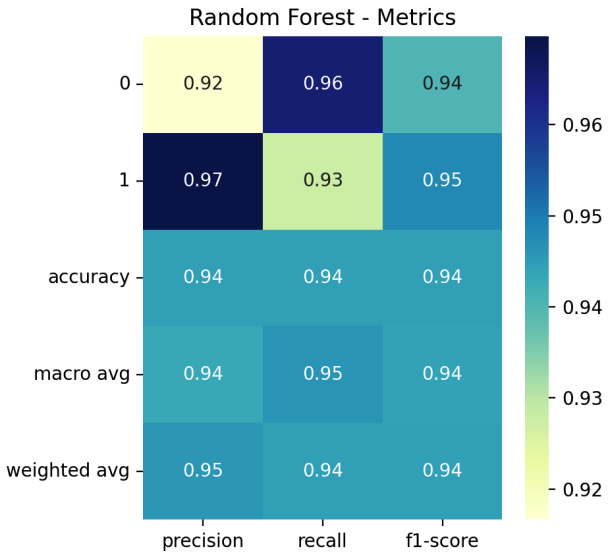

Figure 5.7: Classification Report for Default Random Forest

Classification Report - Tuned Random Forest:

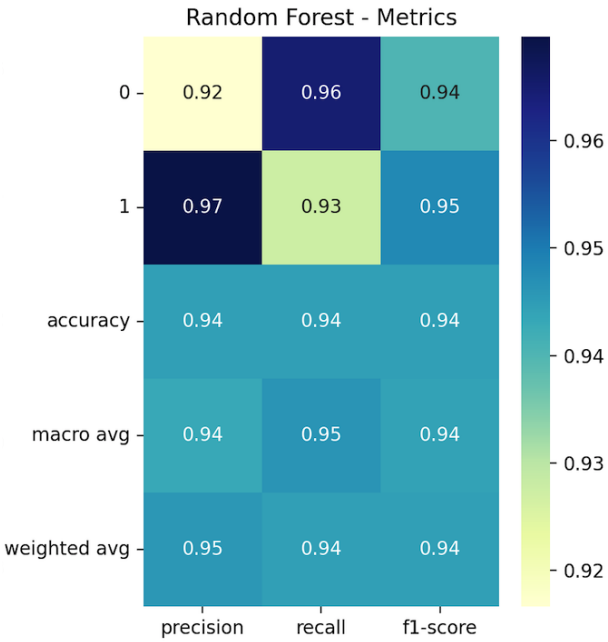

Figure 5.8: Classification Report for Tuned Random Forest

By examining these matrices we can get a deeper understanding of specific types of errors made by each model such as the tendency to misclassify normal eyes as cataract or vice versa. This qualitative evaluation complements the quantitative metrics presented earlier and provides valuable insights into the strengths and weaknesses of each approach.

5.2. Detailed Evaluation Results

```
    def plot_metrics(df_report_linear, df_report_poly,
        df_report_rbf, df_report_rf):
    plt.figure(figsize=(20, 5))

    plt.subplot(1, 4, 1)
    sns.heatmap(df_report_linear[['precision', 'recall', 'f1-
        score']], annot=True, cmap="YlGnBu")
    plt.title('Linear SVM - Metrics')

    plt.subplot(1, 4, 2)
    sns.heatmap(df_report_poly[['precision', 'recall', 'f1-
        score']], annot=True, cmap="YlGnBu")
    plt.title('Polynomial SVM - Metrics')

    plt.subplot(1, 4, 3)
    sns.heatmap(df_report_rbf[['precision', 'recall', 'f1-
        score']], annot=True, cmap="YlGnBu")
    plt.title('RBF SVM - Metrics')

    plt.subplot(1, 4, 4)
    sns.heatmap(df_report_rf[['precision', 'recall', 'f1-
        score']], annot=True, cmap="YlGnBu")
    plt.title('Random Forest - Metrics')

    plt.tight_layout()
    plt.show()
```

5.3 Training and Testing Accuracy Visualization

Moreover to illustrate the performance of the models we visualize the training and testing accuracy for both the default and tuned configurations of each classifier in a single figure for comparison. The visualization provides a concise overview of how well each model learned the training data and its ability to generalize to the unseen testing data. A significant divergence between the training and testing accuracy can indicate overfitting, while closely aligned accuracies suggest good generalization.

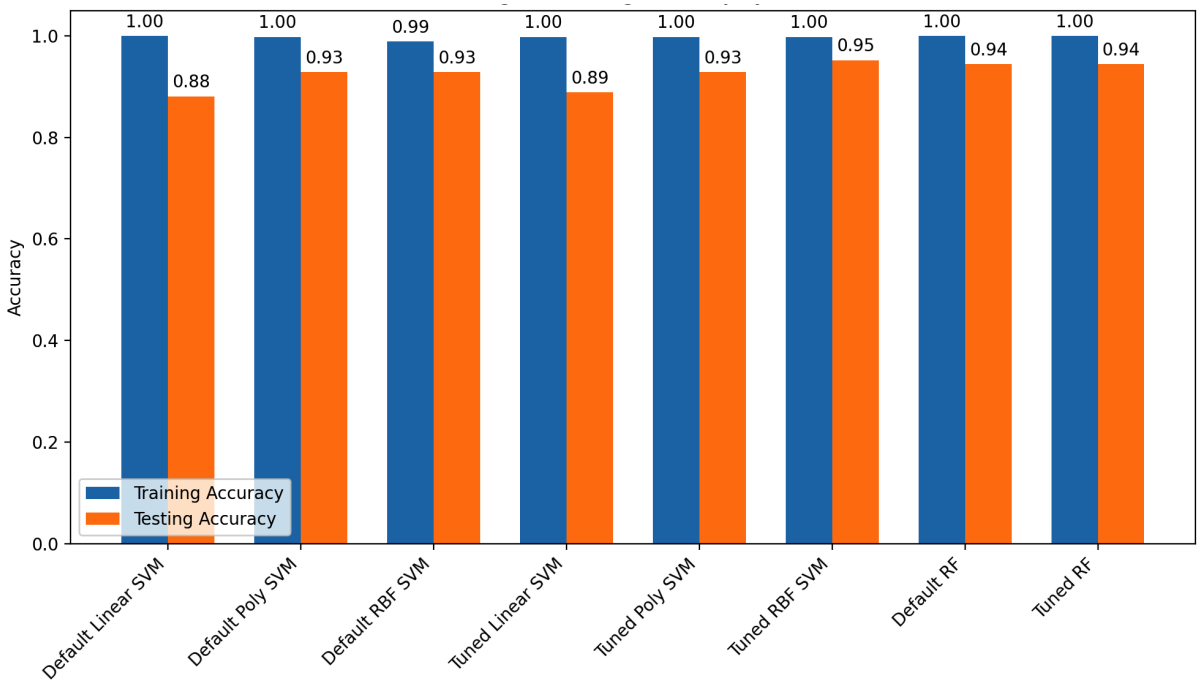

Figure 5.9: Comparison of Training and Testing Accuracy for Default and Tuned Models (All Classifiers).

5.3. Training and Testing Accuracy Visualization

```
def plot_accuracy(train_acc, test_acc, model_names):
    n_models = len(model_names)
    x = np.arange(n_models)
    width = 0.35

    fig, ax = plt.subplots(figsize=(10, 6))
    rects1 = ax.bar(x - width/2, train_acc, width, label='
       Training Accuracy')
    rects2 = ax.bar(x + width/2, test_acc, width, label='
       Testing Accuracy')

    ax.set_ylabel('Accuracy')
    ax.set_title('Training and Testing Accuracy by Model')
    ax.set_xticks(x)
    ax.set_xticklabels(model_names, rotation=45, ha="right")
    ax.legend()

    ax.bar_label(rects1, padding=3, fmt='%.2f')
    ax.bar_label(rects2, padding=3, fmt='%.2f')

    fig.tight_layout()
    plt.show()
```

Chapter 6

Conclusion

This chapter will give the details of the key aspects of research on the cataract detection using machine learning algorithm. We will revisit the objectives of this reserch and summarize our main results and highlight the contributions of our work and discuss its implications and limitations and propose potential avenues for future research and real world applications which include the development of a mobile application.

6.1 Summary of Findings

Our study consists of training, testing and evaluating several machine learning models including Support Vector Machines (SVM) with linear, polynomial and Radial Basis Function (RBF) kernels as well as Random Forest classifier for the binary classification of eye images into 'Normal' and 'Cataract' categories. We assessed the performance of both default and hyperparameter tuned versions of these models on a held out test dataset using various evaluation metrics, including accuracy, precision, recall, F1-score and visualized confusion matrices for a qualitative understanding of the classification errors.

The key findings of our study are as follows:

- **Tuned RBF Support Vector Machine:** achieved the highest overall testing accuracy of 95.24% which is demonstrating the effectiveness of non linear kernel methods for this task when hyperparameters are carefully optimized.
- **Random Forest:** models (both default and tuned) exhibited robust and competitive performance which is achieving testing accuracy of 94.44% and highlighting the strength of ensemble learning techniques.

- **Polynomial SVM:** showed consistent performance of around 92.86% whichi is indicating that the polynomial kernel can also capture relevant non linearities in the data.
- **Linear SVM:** yielded the lowest accuracy (around 88%) suggesting that the relationship between the extracted features and the presence of cataract is likely non linear.
- Hyperparameter tuning played a improtant role in order to improve the performance of RBF SVM underscoring its importance in maximizing the model capabilities.
- Several models exhibited high training accuracies coupled with lower testing accuracies indicating potential overfitting which needs to be addressed in future work.
- Qualitative evaluation through confusion matrix visualization provided valuable insights into specific types of errors made by each model aiding in understanding their strengths and weaknesses.
- As a significant aspect of this research we have taken the initial steps towards translating the developed models into a **prototype mobile application** for accessible cataract screening. ma

6.1.1 Key Contributions

This research contributes to the field of automated image analysis and cataract detection in several ways:

- **Comprehensive Evaluation of Classical ML Models:** We provided a detailed comparative analysis of multiple classical machine learning models (SVM with different kernels and Random Forest) for cataract detection using a specific set of handcrafted features.
- **Demonstration of RBF SVM Efficacy:** Our findings highlight the effectiveness of the RBF kernel in SVM particularly when coupled with hyperparameter tuning, for achieving high accuracy in this classification task.
- **Benchmarking Performance:** The reported accuracies serve as benchmark for future studies utilizing similar datasets and feature extraction techniques, allowing for a comparison of more advanced approaches.
- **Qualitative Error Analysis:** The inclusion of confusion matrix visualizations offers a deeper understanding of the models performance beyond simple numerical metrics aiding in identifying patterns in misclassifications.

- **Insights into Overfitting:** Our analysis of training and testing accuracy discrepancies emphasizes the importance of considering and mitigating overfitting in medical image classification.
- **Development of a Prototype Mobile Application:** A key contribution of this work is the development of a functional prototype mobile application that integrates the trained cataract detection models, demonstrating the potential for real-world deployment and increased accessibility.

6.1.2 Addressing the Research Questions

While the specific research questions were implicitly embedded in the study design our findings directly address the overarching goal of evaluating the feasibility and effectiveness of classical machine learning techniques for automated cataract detection. We successfully demonstrated that models like tuned RBF SVM and Random Forest can achieve high accuracy using a defined set of image features. The comparison across different kernels and the impact of hyperparameter tuning provide answers to questions regarding the optimal model choice and the importance of parameter optimization. The analysis of error patterns through confusion matrices sheds light on the limitations and potential areas for improvement in these approaches. Furthermore the development of a mobile application prototype directly explores the translational aspect of this research into a tangible tool for wider use.

6.1.3 Implications of the Research

The high accuracies achieved by our best performing models which are coupled with the development of a mobile application prototype and have significant implications for the development of accessible and cost effective cataract screening tools. Automated systems based on these techniques especially when deployed on mobile platforms, could potentially:

- **Increase Accessibility:** Provide preliminary screening directly to individuals through their smartphones particularly in areas with limited access to ophthalmologists and heatlcare professionals.
- **Reduce Healthcare Costs:** Offer more affordable alternative to traditional diagnostic methods for initial assessments, reducing the burden on healthcare systems.

- **Enable Early Detection:** Facilitate the identification of cataract through the regular self-screening allowing individuals for timely intervention and potentially preventing vision loss.
- **Assist Clinicians:** Serve as supportive tool for ophthalmologists which have potential for increasing the efficiency and accuracy of diagnoses especially when reviewing large volumes of images.
- **Empower Individuals:** Provide individuals with a convenient and private way to monitor their eye health and seek professional help when necessary.

6.1.4 Limitations of the Study

Our study has several limitations that should be acknowledged:

- **Dataset Size and Diversity:** The dataset used in this study may be sufficient for initial model evaluation but might not fully represent the variability in real world eye images in terms of quality, lighting conditions. The mobile application performance might also be influenced by the diversity of smartphone cameras and user handling.
- **Handcrafted Features:** The reliance on handcrafted features (color means, edge counts, Hu moments) might limit the model ability to capture more subtle and complex visual cues indicative of cataract compared to features learned automatically by deep learning models, especially when applied to images captured under varying conditions by a mobile phone.
- **Binary Classification:** Our study focused on binary classification (Normal vs. Cataract). The mobile application in its current prototype also reflects this. Future development could explore multi class classification to categorize different stages or types of cataracts which would be more clinically relevant.
- **Lack of External Validation:** The models were evaluated on a single held-out test set. The mobile application's performance has not yet been validated on a separate, real-world user base or compared against clinical diagnoses.
- **Computational Resources for Mobile Deployment:** Optimizing the models for efficient deployment on mobile devices with limited computational resources might require further investigation and model compression techniques.
- **User Interface and Usability:** The current mobile application is a prototype and its user interface and overall usability would need to be rigorously tested and refined based on user feedback.

6.2 Future Work

Building upon the findings and acknowledging the limitations of this study several avenues for future research and development can be explored particularly concerning the mobile application:

6.2.1 Potential Extensions of the Current Work

- **Integration of Deep Learning Models in the Mobile App:** Exploring the feasibility of integrating more advanced deep learning models, potentially optimized for mobile devices, into the application to improve accuracy.
- **Real-time Analysis:** Optimizing the image processing and classification pipeline for real-time analysis within the mobile application.
- **User Feedback and Data Collection:** Implementing mechanisms within the mobile application to collect user feedback and potentially anonymized image data to further train and refine the models.**
- **Integration with Telehealth Platforms:** Exploring the potential integration of the mobile application with telehealth platforms to facilitate remote consultations with ophthalmologists.
- **Severity Assessment in the Mobile App:** Expanding the application's capabilities to not only detect cataract but also to provide an initial assessment of its severity.

6.2.2 Suggestions for Future Research Directions

- **Mobile Specific Data Augmentation:** Investigating data augmentation techniques specifically tailored to variations in images captured by smartphone cameras.
- **Federated Learning for Mobile Data:** Exploring federated learning approaches to train models on data collected from multiple mobile devices while preserving user privacy.
- **Usability Studies for the Mobile App:** Conducting rigorous usability studies with target users to optimize the application's interface and workflow.
- **Clinical Validation of the Mobile App:** Conducting clinical studies to validate the accuracy and reliability of the mobile application against traditional diagnostic methods.

6.2.3 Addressing the Limitations

Future work should specifically aim to address the limitations identified in this study and the mobile application by:

- **Large Diverse Mobile Image Datasets:** Collecting and gathering larger diverse dataset of eye images captured using various smartphone, under different real world conditions.
- **Robustness Testing on Mobile Images:** Rigorously testing the models' performance on images captured by mobile application under different lighting, focus and user handling conditions.
- **Comparative Study on Mobile Platform:** Comparing the performance of traditional ML models with optimized deep learning model when deployed on mobile devices.
- **User Centered Design and Testing:** Iteratively designing and testing the application user interface based on feedback from users and healthcare professionals.

6.2.4 Exploring Alternative Approaches

For mobile based solutions future research could also explore:

- **AI on Edge Computing:** Optimizing model deployment for on the device inference to minimize latency and reliance on network connectivity.
- **Interactive Guidance for Image Capture:** Implementing features with in the mobile application to guide user about how to capture high quality eye images suitable for cataract analysis.

6.2.5 Real world Applications and Impact

The development of functional mobile application based on our research have a significant promise for real world impact by increasing the accessibility of cataract screening particularly in underserved communities. Future efforts will focus on rigorous validation, user testing and refinement to make sure the reliability and usability of application as potential tool for early cataract detection and management. Successful deployment could contribute significantly to reducing the burden of cataract related blindness globally.

6.2. Future Work

In wrap the things up our study provides valuable contribution to the field of automated cataract detection and culminating in the development of prototype mobile application. The results achieved with our machine learning techniques lay solid foundation for future research aimed by developing more accurate, robust and accessible tools including mobile solutions for combating cataract related blindness worldwide.

References

[1] Vaibhav Agarwal, Vaibhav Gupta, Vivasvan Manasvi Vashisht, Kiran Sharma, and Neetu Sharma. Mobile application based cataract detection system. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 780–787, 2019.

[2] Manoj Kumar Behera, S. Chakravarty, Apurwa Gourav, and Satyabrata Dash. Detection of nuclear cataract in retinal fundus image using radialbasis functionbasedsvm. In *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pages 278–281, 2020.

[3] Yanyan Dong, Qinyan Zhang, Zhiqiang Qiao, and Ji-Jiang Yang. Classification of cataract fundus image based on deep learning. In *2017 IEEE international conference on imaging systems and techniques (IST)*, pages 1–5. IEEE, 2017.

[4] Yunendah Nur Fuadah, Agung W. Setiawan, Tati L.R. Mengko, and Budiman. Mobile cataract detection using optimal combination of statistical texture analysis. In *2015 4th International Conference on Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME)*, pages 232–236, 2015.

[5] Weihao Gao, Lei Shao, Fang Li, Li Dong, Chuan Zhang, Zhuo Deng, Peiwu Qin, Wenbin Wei, and Lan Ma. Fundus photograph-based cataract evaluation network using deep learning. *Frontiers in Physics*, 11, 2024.

[6] Ishitaa Jindal, Palak Gupta, and Anmolika Goyal. Cataract detection using digital image processing. In *2019 Global Conference for Advancement in Technology (GCAT)*, pages 1–4, 2019.

[7] Masum Shah Junayed, Md Baharul Islam, Arezoo Sadeghzadeh, and Saimunur Rahman. Cataractnet: An automated cataract detection system using deep learning for fundus images. *IEEE Access*, 9:128799–128808, 2021.

[8] Ahmed Tuama Khalaf and Salwa Khalid Abdulateef. Ophthalmic diseases classification based on yolov8. *Journal of Robotics and Control (JRC)*, 5(2), 2024.

[9] Raghu Nandhan Meegada, Ashwan Lingala, Vivek Mamidi, and Sai Abhinav Bharadwaj. Efficient detection of eye diseases using ml and dl. Technical report, EasyChair, 2024.

[10] Nandan P. Cataract image dataset. https://www.kaggle.com/datasets/nandanp6/cataract-image-dataset, 2025. Accessed: March 24, 2025.

[11] Flutter Packages. Flutter packages repository. https://pub.dev/, 2025. Accessed: March 24, 2025.

[12] Akshay Bhuvaneswari Ramakrishnan, Mukunth Madavan, R Manikandan, and Amir H Gandomi. A hybrid deep learning paradigm for robust feature extraction and classification for cataracts. *Applied AI Letters*, 6(2):e113, 2025.

[13] Juyel Rana and Syed Md. Galib. Cataract detection using smartphone. In *2017 3rd International Conference on Electrical Information and Communication Technology (EICT)*, pages 1–4, 2017.

[14] Nihar Ranjan, Rohan Haral2 Atharva Shejul, Kinjal Harne, and Shravan Bhat. Detection of cataract and its level based on deep learning using mobile application. 2023.

[15] Flutter Team. Flutter official documentation. https://flutter.dev/, 2025. Accessed: March 24, 2025.

[16] Chandrakumar Subbiah Vasan, Sachin Gupta, Madhu Shekhar, Kamatchi Nagu, Logesh Balakrishnan, Ravilla D Ravindran, Thulasiraj Ravilla, and Ganesh-Babu Balu Subburaman. Accuracy of an artificial intelligence-based mobile application for detecting cataracts: Results from a field study. *Indian Journal of Ophthalmology*, 71(8):2984–2989, 2023.

[17] Mano Aarthi VM, Nivedita Tiwari, Vinay Khobragade, Mitali Pareek, Anand Panchbhai, Priyanjit Ghosh, and Jayachandhran Saravanan. Clinical validation of artificial intelligence-based cataract screening solution with smartphone images (logy ai cataract screening module). *International Journal of Advances in Medicine*, 11(2):71, 2024.

[18] Meimei Yang, Ji-Jiang Yang, Qinyan Zhang, Yu Niu, and Jianqiang Li. Classification of retinal image for automatic cataract detection. In *2013 IEEE 15th International Conference on e-Health Networking, Applications and Services (Healthcom 2013)*, pages 674–679, 2013.