TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A REPORT ON
IRIS RECOGNITION SYSTEM USING
SIAMESE NETWORK

SUBMITTED BY:

HIMANSHU PRADHAN (PUL077BCT030)
JANAM SHRESTHA (PUL077BCT032)
MEJAN LAMICHHANE (PUL077BCT047)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

March, 2024

Copyright

Acknowledgement

We express our sincere gratitude to the Department of Electronics and Computer Engineering for providing us with the opportunity and steadfast encouragement in the development our minor project, Iris Recognition System employing Siamese Neural Network. This collaborative effort has been enriched by the resources and guidance provided by the Professors and peers throughout the conceptualization and planning phases.

First and foremost, we would like to express our deepest appreciation to **Prof. Dr. Sanjib Prasad Pandey**, our project supervisor from the Department of Electronics and Computer Engineering, for their invaluable mentorship, insightful feedback, and continuous encouragement. Their expertise and guidance have been instrumental in shaping the direction and methodology outlined in this proposal. We express our deepest appreciation to Prof. Dr. Jyoti Tandukar, the Head of the Department, for their visionary leadership and encouragement, which laid the foundation for our project's success. We also thank Professor Santosh Giri for his invaluable guidance and support, contributing significantly to the project's advancement.

We are also thankful to the faculty and staff of the Department of Electronics and Computer Engineering at **Institute of Engineering, Pulchowk Campus** for fostering a conducive learning environment and providing the necessary resources for the development of this proposal.

Furthermore, we extend our appreciation to our peers and colleagues within the department who have shared their knowledge and offered assistance at various stages of crafting this proposal. Their guidance played a pivotal role in refining our approach and achieving significant progress.

Abstract

Iris Recongnititon System using Siamese Neural Network Architecture is a biometric authentication system that uses neural network for iris comparison and authentication The system is based on an empirical analysis of the iris image and it is split in several steps using local image properties. The system steps are capturing iris patterns; determine the location of the iris boundaries; converting the iris boundary to the stretched polar coordinate system; segmentation and normalization of the iris; feature extraction and similarity score comparison using SNN. The proposed system uses Circular Hough transformation for pupil localization. Subsequently, it employs a differential intensity approach to accurately identify the boundaries of the iris. Finally, Sobel filtering is utilized to effectively detect the edges of the eyelids. The system was implemented using a dataset obtained from IIT Delhi Iris dataset comprising of 2,240 iris images from 224 different individuals and MMU Iris Dataset that had 460 images from 46 different individuals.

Keywords: Iris Recognition, Siamese Neural Network, Biometric Authentication, Neural Network, Iris Comparison, Authentication, Empirical Analysis, Iris Image, Local Image Properties, Circular Hough Transformation, Pupil Localization, Differential Intensity Approach, Iris Boundary Detection, Sobel Filtering, Eyelid Detection, Dataset, IIT Delhi Iris Dataset, MMU Iris Dataset.

List of Abbreviations

CNN	Convolutional Neural Network
IIT	Indian Institue of Technology
MMU	Multimedia University
NIR	Near-Infrared Light
NN	Neural Network
ReLu	Rectified Linear Unit
ResNet	Residual Network
SNN	Siamese Neural Network
TF	Tensorflow
VGG	Visual Geometry Group
CSV	Comma Separated File
Adam	Adaptive Moment Estimation
AdaGrad	Adaptive Gradient Descent
RMSprop	Root Mean Squared Propagation
SNN	Siamese Neural Network

Contents

List of Figures

1. Introduction

Biometric authentication systems have revolutionized the landscape of security and access control, offering unprecedented levels of accuracy and reliability. Iris recognition, specifically relies on the unique patterns found within the iris to distinguish between individuals. It functions akin to a sophisticated security mechanism capable of identifying individuals by simply analyzing their eyes. Renowned for its high accuracy and resistance to deception, this technology serves a variety of purposes, from unlocking smartphones to granting access to secure locations or verifying identities for crucial activities. As advancements in technology continue, iris recognition is increasingly adopted in diverse settings to bolster safety and security measures.

Our project, titled "Iris Recognition Using Siamese Neural Network Architecture," embodies a focused initiative dedicated to advancing the functionalities of iris recognition systems through inventive methodologies and state-of-the-art neural network architectures.

1.1 Background

The human iris, with its unique patterns, offers a fascinating solution for biometric authentication. We explored the biological foundation of this technology, where melanin pigments within the iris, eumelanin and pheomelanin, create intricate patterns due to their interaction with light. These patterns remain remarkably stable throughout a person's life, making the iris an ideal anatomical feature for identification.

Now, let's delve into the technical aspects of iris recognition. Specialized cameras capture high-resolution images of the iris, often utilizing both visible and near-infrared (NIR) light. NIR light penetrates deeper, revealing intricate details invisible to the naked eye. Specific lighting conditions might also be used to optimize image capture.

Once captured, image processing algorithms take center stage. The iris region is meticulously isolated from surrounding areas like eyelashes and the sclera (white of the eye). Specialized algorithms then extract unique features from the segmented iris image. These features can include the intricate network of connective tissue (stroma), tiny lace-like structures (crypts), radiating lines that appear during pupil constriction (furrows), and even concentric bands of color (pigmented rings). These extracted features are then used to create a digital template, a unique iris code for each individual, which is securely stored in a database.

During authentication, a captured iris image is compared against these stored templates.

Matching algorithms analyze the similarity between the features extracted from the new image and those stored in the database. A successful match verifies the claimed identity.

The advantages of iris recognition are compelling. The unique patterns on the iris offer a high degree of distinction, even between identical twins. Additionally, the iris remains relatively unchanged throughout life, making it a reliable identifier. The contactless nature of iris capture provides user convenience and hygiene benefits. Finally, the iris data resides within the protected environment of the eye, making it difficult to forge or copy, enhancing security.

The applications of iris recognition technology are vast. From verifying traveler identities at borders and aiding law enforcement in criminal identification to securing access to financial services and controlling entry to restricted areas, iris recognition offers a powerful tool for identification and authentication.

Looking ahead, advancements in camera technology and algorithms promise to further improve capture speed and accuracy. The integration of iris recognition into mobile devices is a growing trend, offering convenient and secure authentication on the go. Combining iris recognition with other biometric modalities like fingerprint or facial recognition can create even more robust security solutions. As technology evolves, iris recognition has the potential to become an even more ubiquitous and reliable method for safeguarding our identities and controlling access.

1.2 Problem Statement

Developing robust and secure biometric authentication systems remains a crucial challenge. Traditional methods like passwords and tokens are susceptible to theft or loss. This necessitates exploring alternative solutions that leverage unique and stable physiological characteristics for reliable user identification.

In this context, iris recognition presents a promising approach. However, limitations exist in terms of capture speed, accuracy under varying lighting conditions, and potential biases within the matching algorithms. Additionally, ensuring user privacy and data security during iris template storage and retrieval requires careful consideration.

The problem lies in designing and implementing an iris recognition system that addresses these limitations. The system should achieve high accuracy and speed while operating under diverse lighting conditions. Furthermore, it must be robust against potential biases and ensure user privacy through secure data storage and handling practices.

1.3 Objectives

- Develop a high-accuracy iris recognition system that reliably identifies individuals.

- Enhance capture speed to facilitate user convenience and minimize wait times.
- Explore algorithms robust against variations in lighting conditions, ensuring consistent performance.
- Mitigate potential biases within the matching algorithms to achieve fair and unbiased identification.
- Implement secure data storage and handling practices to protect user privacy and prevent unauthorized access to iris templates.

1.4 Scope

This project focuses on developing a core iris recognition system that enables user enrollment and identity verification through iris scans. The project deliverables will include:

- Functional Iris Recognition System: This system will allow users to enroll by capturing their iris images and then utilize those images for verification purposes.
- Image Segmentation: This module will isolate the iris region from surrounding areas like eyelashes and eyelids within the captured image.
- Feature Extraction: This module will extract unique and stable features from the segmented iris image, forming the basis for identification.
- Matching Algorithm: A core component, this algorithm will compare the features extracted from a new iris scan against stored templates during verification, determining a match or non-match.
- Secure Template Storage: A secure mechanism will be implemented to store user iris templates, protecting this sensitive data.

However, certain aspects are excluded from this project scope:

- Integration with Existing Systems: Integration with existing security systems or databases is outside the project's scope.
- Deployment and Maintenance: Deploying the system in a real-world environment and its ongoing maintenance are not included.
- Legal and Ethical Considerations: Addressing legal and ethical concerns surrounding iris data collection and storage falls outside this project's scope.

2. Literature Review

2.1 Related work

John Daugman's groundbreaking research in iris recognition, particularly his development of Gabor's phase-quadrant feature descriptors, played a pivotal role in its success back in 2002. His contributions paved the way for the creation of precise algorithms that focused on crucial aspects such as segmentation, normalization, feature extraction, and template classification. The integration of Deep Learning into iris recognition emerged later, around 2016, largely due to various challenges, including the demand for extensive training data and limited exploration. Noteworthy applications of Deep Learning in iris recognition include the development of DeepIrisNet and studies conducted by Nguyen et al. and Al-Waisy et al., where Convolutional Neural Networks (CNNs) were utilized for feature extraction and classification.

In a separate study [2], researchers proposed a robust system for face recognition utilizing a robust 4-layer CNN architecture capable of handling facial images with diverse characteristics such as facial expressions, poses, occlusions, and varying illumination conditions. Their experiments demonstrated a high accuracy rate of 99.5

Another study [3] proposed a system for diagnosing iris nevus, a pigmented growth around the pupil or in the front of the eye, using a combination of Convolutional Neural Network and deep belief network. They employed a pre-trained LeNet-5 architecture for CNN and achieved accuracy rates of 93.35

In a different approach [4], researchers proposed an iris recognition system that utilized a pre-trained VGG-Net for extracting deep features. They then employed a multi-class SVM algorithm for classification and evaluated their system on two iris databases, IIT iris dataset, and CASIA 1000 Iris dataset, achieving a high recognition accuracy of 99.4

In contrast to previous iris recognition systems that were primarily confined to CNN usage, our proposed approach involves leveraging similarity learning through a Siamese Network. This innovative methodology aims to overcome the performance limitations observed in earlier works, promising enhanced efficiency and effectiveness in the recognition process.

2.2 Related theory

2.2.1 Gaussian Blur

Gaussian blur is a widely used technique in image processing and computer vision for reducing image noise and smoothing out details. It is based on the convolution of the image with a Gaussian kernel, which is a 2D bell-shaped function defined by its standard deviation (σ). The Gaussian kernel emphasizes pixels closer to the center more than those farther away, resulting in a smooth transition of pixel values across the image. This process effectively blurs the image while preserving its overall structure and edges.

Mathematically, the Gaussian blur operation can be represented as a convolution between the input image I and the Gaussian kernel G:

$$\text{Blurred image} = I * G$$

Where $*$ denotes the convolution operation. The Gaussian kernel is defined as:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where x and y are the spatial coordinates, and σ is the standard deviation of the Gaussian distribution. The larger the value of σ, the more pronounced the blurring effect will be.

2.2.2 Median Blur

Median blur is a popular image processing technique used for reducing noise and smoothing out images. Unlike Gaussian blur, which applies a weighted average to the pixels in a neighborhood, median blur replaces each pixel's value with the median value of its neighboring pixels. This approach is particularly effective at preserving edges and fine details while removing noise.

Mathematically, the median blur operation involves sliding a window over the image and replacing each pixel's value with the median value of the pixel values within the window. The size of the window determines the extent of blurring, with larger windows resulting in more pronounced smoothing effects.

Median blur is robust to outliers and is especially useful for removing salt-and-pepper noise, where isolated pixels have extremely high or low intensity values. It is commonly used in various image processing tasks, including denoising, edge preservation, and feature extraction.

Unlike Gaussian blur, which can produce a more gradual and isotropic blur, median blur tends to produce sharper transitions between regions of different intensities. This charac-

teristic makes it particularly suitable for preserving edges and details in the image while reducing noise.

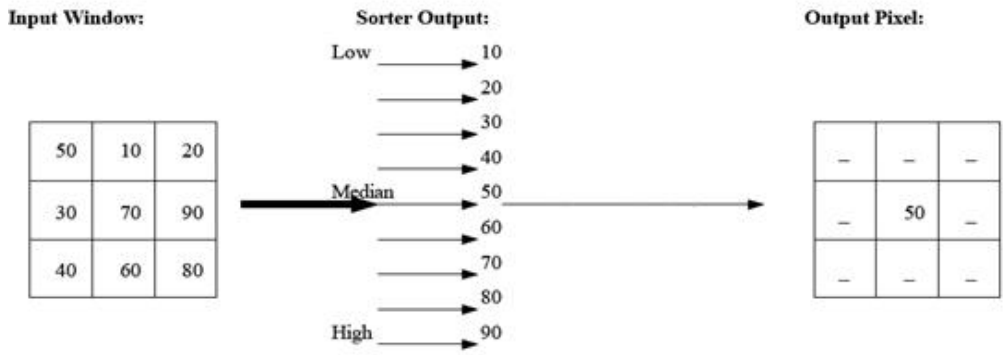


Figure 2.1: Median Blur

2.2.3 Hough Transform

The Hough Transform is a fundamental technique in image processing and computer vision used for detecting geometric shapes, particularly lines and circles, in an image. It works by transforming the image space into parameter space, where each point in the image corresponds to a curve in parameter space. This transformation allows for the detection of lines or curves by identifying the points in parameter space that intersect the most curves, indicating the presence of a particular geometric shape.

In the context of line detection, the Hough Transform represents each point in the image space as a line in parameter space, typically using the slope-intercept form (Hough space). By accumulating votes for potential lines in parameter space, lines in the image space can be detected as peaks in the Hough space.

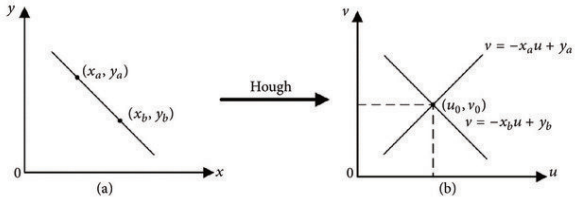
Similarly, for circle detection, the Hough Transform represents each point in the image space as a circle in parameter space, typically using the center coordinates and radius (Hough circle space). By accumulating votes for potential circles in parameter space, circles in the image space can be detected as peaks in the Hough circle space.

The Hough Transform has numerous applications in image analysis, including edge detection, shape recognition, and object tracking.

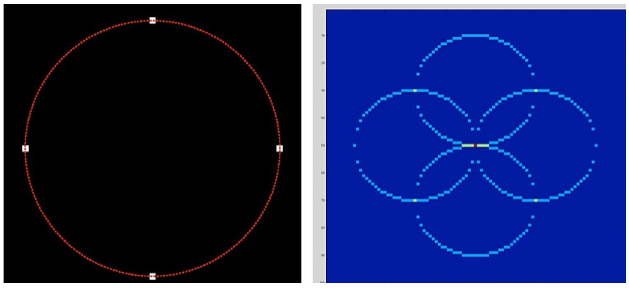
Circular Hough Transform Algorithm

1. Determine the range of ρ and θ.

2. Create a 3D array called the accumulator with dimensions (num_rhos, num_thetas, num_radii) to represent the Hough Space for circles and set all its values to zero.



(a) Hough Line Transform



(b) Hough Circle Transform

Figure 2.2: Different Hough Transforms

3. Use the original Image to perform edge detection (ED). This can be done using any edge detection technique.

4. Check each pixel on the edge picture to determine if it is an edge pixel. If the pixel is on the edge, loop over all possible values of θ and ρ, compute the corresponding ρ, θ, and radius, locate the θ, ρ, and radius index in the accumulator, and then increment the accumulator based on those index triplets.

5. Iterate over the values in the accumulator. Retrieve the ρ, θ, and radius index and obtain the value of ρ, θ, and radius from the index triplet. These values may then be used to represent the detected circles.

2.2.4 Sobel Filter

The Sobel filter is a popular edge detection technique used in image processing to identify edges in an image. It works by calculating the gradient of the image intensity at each pixel using convolution with two 3x3 kernels: one for horizontal changes and the other for vertical changes.

The horizontal Sobel kernel is:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

And the vertical Sobel kernel is:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

To apply the Sobel filter, these kernels are convolved with the input image to compute the gradient in the horizontal and vertical directions. The gradient magnitude at each pixel is then calculated as the square root of the sum of the squares of the horizontal and vertical gradients.

The resulting gradient magnitude image highlights edges in the original image, with stronger responses at locations where the intensity changes rapidly. This makes the Sobel filter a valuable tool for various image processing tasks, including edge detection, feature extraction, and object recognition.

2.2.5 Rubbersheet Mapping

Rubbersheet mapping is a technique commonly used in iris recognition for transforming the iris texture into a more uniform representation. This process involves mapping the iris texture from Cartesian coordinates to polar coordinates.

In rubbersheet mapping for iris recognition, the iris region is first localized within the input image. Subsequently, the iris texture is extracted and mapped onto a polar grid, where the radial direction represents the distance from the pupil center, and the angular direction represents the orientation around the pupil.

By performing rubbersheet mapping, the iris texture is normalized and aligned, making it more invariant to changes in pupil dilation and iris rotation. This transformation enhances the performance of iris recognition algorithms by reducing the variability introduced by different imaging conditions.

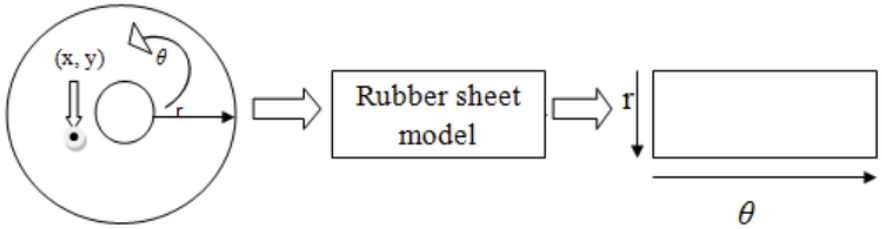


Figure 2.3: Rubersheet Mapping

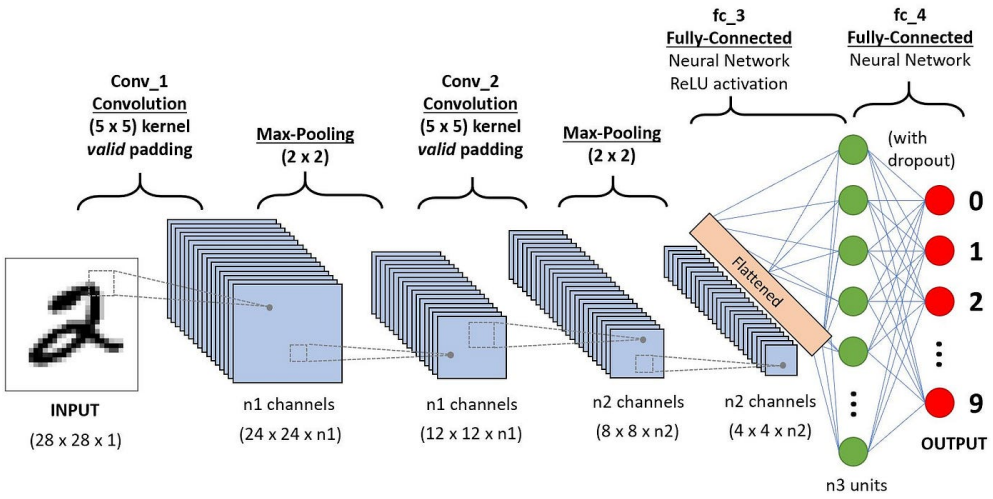


Figure 2.4: CNN architecture

2.2.6 Convolutional Neural Network

Need of CNN

One of the largest limitations of traditional forms of ANN is that they tend to struggle with the computational complexity required to compute image data. If we consider coloured image input of 64×64, the number of weights on just a single neuron of the first layer increases substantially to $12,288$. This problem cannot be solved by increasing the number of hidden layers or increasing neurons because it would result in very **high computational** cost **overfitting**.

To solve these problems, CNN was developed. CNN is a type of deep learning model for processing data that has a grid pattern, such as images, which is inspired by the organization of animal visual cortex and designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns. CNN is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers.

Convolution and Convolution Layer

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function. **Convolution:** Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a tensor. An

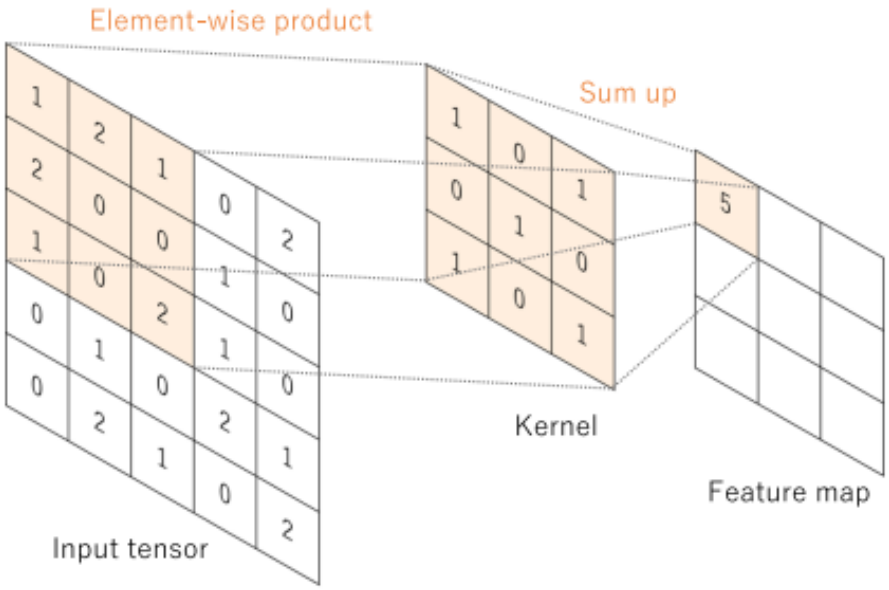


Figure 2.5: An example of convolution operation with a kernel size of 3 × 3, no padding, and a stride of 1.

element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map. Two key hyperparameters that define the convolution operation are size and number of **kernels**. The former is typically 3 × 3. Also, the **stride** is the step size with which the kernel moves across the input data. When convolution operation is applied to input feature tensor, the size of the feature maps may be smaller than the input size, especially at the edges. **Padding** addresses this issue by adding extra pixels around the input feature maps before applying the convolution operation.

Activation: The outputs of a linear operation such as convolution are then passed through a nonlinear activation function. Non-linear activation functions enable the network to learn complex, non-linear relationships between features in the data, which is essential for capturing intricate patterns and representations. The most common nonlinear activation function used presently is the rectified linear unit (ReLU), which simply computes the function: f(x) = max(0, x).

Pooling and Pooling Layer

Pooling layers aim to gradually reduce the dimensionality of the representation, and thus further reduce the number of parameters and the computational complexity of the model. The most popular form of pooling operation is **max pooling**, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other

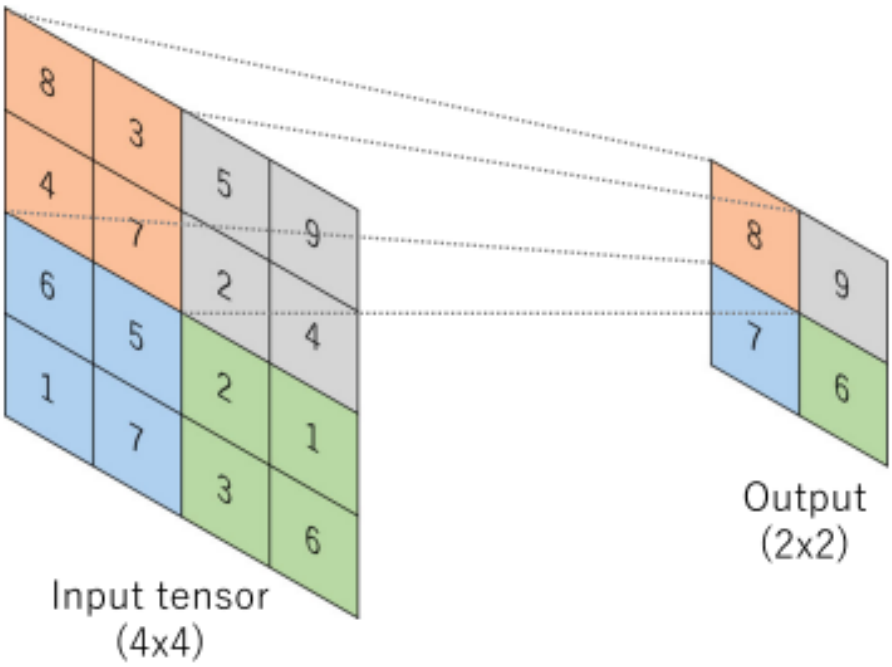


Figure 2.6: An example of max pooling operation with a filter size of 2 × 2, no padding, and a stride of 2.

values.

Fully connected layer

The output feature maps of the final convolution or pooling layer is typically flattened, i.e., transformed into a one-dimensional (1D) array of numbers (or vector), and connected to one or more fully connected layers, also known as dense layers, in which every input is connected to every output by a learnable weight.

2.2.7 Siamese Networks

The need for Siamese Networks emerged from the limitations of traditional classification methods in discerning relationships between objects or patterns that are not easily distinguishable through straightforward classification techniques. Siamese Networks overcome these challenges by learning a robust representation of input pairs and generalize which images are similar and dissimilar.

A Siamese Neural Network is a type of neural network architecture designed to learn similarity or dissimilarity between pairs of inputs. It consists of two identical sub-networks (often called twins or branches) with shared weights, each processing one of the input instances independently. The Layers of Siamese Network are as follows:

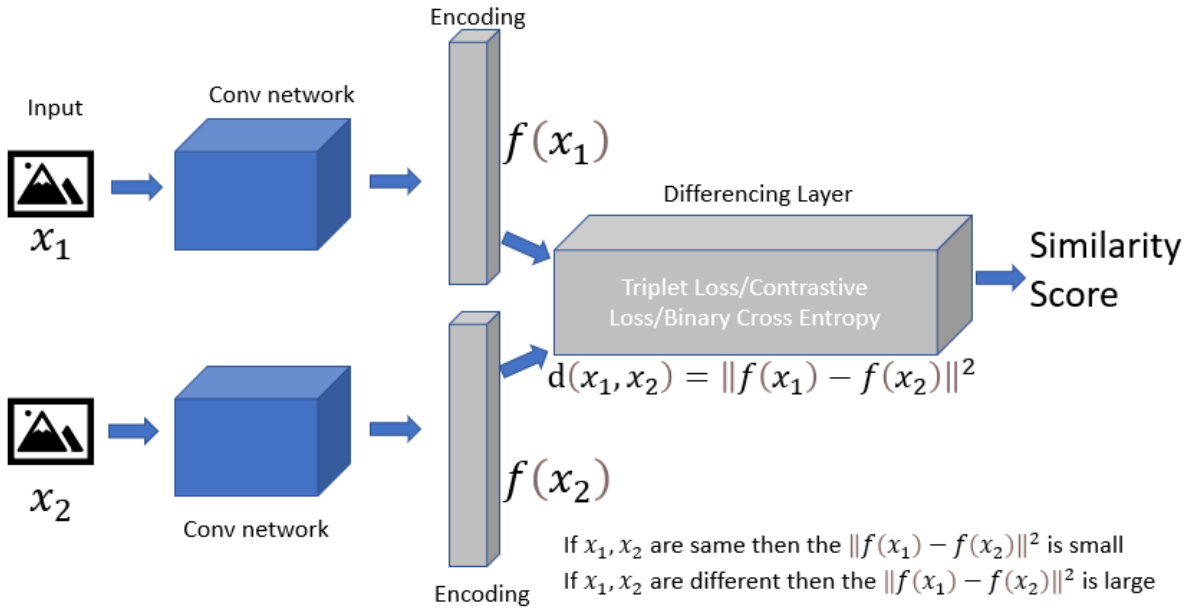


Figure 2.7: Siamese Network Illustration

- Input Layer: This layer receives the input data, which could be images, text, or any other type of data.
- Feature Extraction/Encoding Layers: These layers extract meaningful features from the input data. Commonly used feature extraction layers include convolutional layers (for images) or recurrent layers (for sequential data like text).
- Flatten Layer: This layer converts the multi-dimensional output of the feature extraction layers into a one-dimensional vector, which can be fed into subsequent layers.
- Differencing Layer: This layer calculates the distance between flattened feature vector of two input images.
- Output Layer: In Siamese networks used for similarity tasks, the output layer typically consists of a single neuron that computes the similarity score between the input images using the output of differencing layer. Common activation functions for this layer include sigmoid or cosine similarity.

3. Methodology

The goal of the Iris Recognition System is to utilize sophisticated computer vision methods and machine learning algorithms to accomplish precise and secure iris-based verification. The suggested approach entails a structured sequence of actions, including gathering data, preparing it, constructing models, designing web applications, and integrating systems. The subsequent sections elaborate on each phase of the approach, offering a thorough explanation of the procedures that will be utilized to achieve the project's objectives.

3.1 Feasibility Study

3.1.1 Technical Feasibility

The technical feasibility of the iris recognition system is grounded in the utilization of well-established methodologies and technologies across various stages of the system pipeline. At the front of this system lies the preprocessing stage, where techniques such as noise reduction, image enhancement, and normalization are employed to ensure the input iris images are properly prepared for subsequent analysis. These preprocessing steps are supported by a wealth of research and practical applications, making them reliable and effective in real-world scenarios.

Moving beyond preprocessing, the system leverages sophisticated algorithms for feature extraction. Here, the use of neural network architectures, particularly Convolutional Neural Networks (CNNs), proves invaluable. CNNs excel in tasks involving image analysis and pattern recognition, making them well-suited for the intricate task of iris detection. By employing CNNs, the system can accurately identify and localize iris patterns within input images, laying the foundation for subsequent authentication processes.

Moreover, the feasibility of the system is supported by the availability of comprehensive datasets for training and validation purposes. These datasets provide ample examples of iris images captured under various conditions, allowing the system to learn and generalize effectively. Additionally, the adoption of deep learning frameworks such as TensorFlow and PyTorch facilitates the implementation and deployment of complex neural network architectures, further enhancing the system's feasibility.

3.1.2 Economic Feasibility

The economic feasibility of the iris recognition system is supported by several factors that contribute to cost-effectiveness and affordability throughout the system's development and deployment phases. One of the primary cost-saving measures stems from the availability of Graphics Processing Units (GPUs), which serve as valuable resources for training and executing complex neural network models. While dedicated GPU hardware can incur significant upfront costs, alternative options such as cloud-based platforms like Google Colab offer accessible and cost-effective alternatives. By leveraging cloud-based GPU resources, developers can access high-performance computing capabilities without the need for substantial investments in hardware infrastructure. This flexibility in resource allocation allows for efficient utilization of computational resources, optimizing costs while ensuring the timely completion of model training and evaluation tasks.

Furthermore, the economic feasibility of the system is supported by the availability of freely accessible iris datasets from sources like IIT Delhi and Multimedia University, which serve as crucial resources for model development and validation. By leveraging these publicly available datasets, developers costs mitigate associated with dataset procurement can be mitigated.

Additionally, the open-source nature of many deep learning frameworks and libraries further contributes to the economic viability of the system. Platforms such as TensorFlow, PyTorch, and Keras provide powerful tools and resources for building and deploying neural network models, at no cost.

3.2 Requirement Analysis

The following are the expected requirements of the system.

3.2.1 Functional Requirements

The system is expected to fulfill several functional requirements to ensure its effectiveness in iris recognition and authentication. These requirements encompass various aspects of image processing, authentication procedures, database management, and user interaction. Below are the detailed functional requirements:

- **Input Image Processing**: The system should be capable of accepting input images containing iris patterns captured under different lighting conditions and angles.
- **Preprocessing Tasks**: It should perform preprocessing tasks to enhance the quality of input images and remove any noise or artifacts that may interfere with accurate iris recognition.

- **Segmentation Techniques**: Application of segmentation techniques to isolate and extract the iris region from the input images, ensuring precise identification of iris patterns.
- **User Authentication**: Authentication of users based on comparison with pre-enrolled iris templates stored securely in the system's database. Only users whose iris patterns match the stored templates should be granted access.
- **Matching Algorithms**: Utilization of robust matching algorithms such as Convolutional Neural Networks (CNNs) or Siamese Neural Networks (SNNs) for comparing input iris patterns with the stored templates to determine authentication.
- **Database Management**: Management of a secure database to store pre-enrolled iris templates and associated user information. The database should ensure data integrity, confidentiality, and accessibility.
- **Multiple Iris Images**: Capture and storage of multiple iris images per user to accommodate variations in iris patterns due to factors like lighting conditions, aging, or occlusions.
- **Access Controls**: Implementation of access control mechanisms to restrict unauthorized access to sensitive user data and ensure compliance with privacy regulations.
- **Encryption Mechanisms**: Integration of encryption techniques to safeguard stored data, including iris templates and user credentials, from unauthorized access or tampering.
- **Error Handling**: Handling of authentication failures with informative error messages to guide users on potential corrective actions or alternative authentication methods.

3.2.2 Non-Functional Requirements

In addition to functional capabilities, the system must adhere to various non-functional requirements to meet user expectations and operational standards. These non-functional requirements encompass performance, scalability, user experience, and other crucial aspects. Below are the detailed non-functional requirements:

- **Accuracy**: The system must achieve a high accuracy rate in iris recognition and authentication tasks, minimizing false acceptance and rejection rates to ensure reliable user verification.

- **Processing Speed**: It should exhibit fast processing speeds to perform authentication and verification tasks swiftly and efficiently, minimizing user wait times and maximizing system responsiveness.
- **Scalability**: The system should be scalable to accommodate increasing user loads and expanding datasets without compromising performance or functionality. It should handle concurrent user requests effectively.
- **Reliability**: Ensuring system reliability by minimizing downtime and errors, maintaining consistent performance levels, and providing mechanisms for fault tolerance and recovery in case of failures.
- **Security**: Implementation of security measures to protect sensitive user data, including iris templates, user credentials, and system configurations, from unauthorized access, manipulation, or breaches.
- **Usability**: Offering an intuitive and user-friendly interface for easy interaction with the system, facilitating smooth navigation, clear feedback, and minimal learning curve for users of varying technical proficiency.
- **Accessibility**: Ensuring accessibility for users with disabilities by providing support for assistive technologies, adherence to accessibility standards, and consideration of diverse user needs in interface design.
- **Compatibility**: Compatibility with a wide range of hardware and software environments, including different operating systems, browsers, and device types, to maximize system accessibility and usability.
- **Maintainability**: Designing the system with maintainability in mind, including modular architecture, well-documented code, version control practices, and automated testing procedures to facilitate future updates, enhancements, and troubleshooting.
- **Performance Optimization**: Continuously optimizing system performance through efficient resource utilization, algorithmic improvements, caching strategies, and other techniques to enhance overall efficiency and responsiveness.

3.3 Data Collection

The acquisition of image data for this study involved sourcing from two primary datasets: the MMU (Multimedia University) and IIT Delhi (Indian Institute of Technology Delhi) datasets. From the MMU dataset, a total of 460 images were obtained, originating from 46 distinct

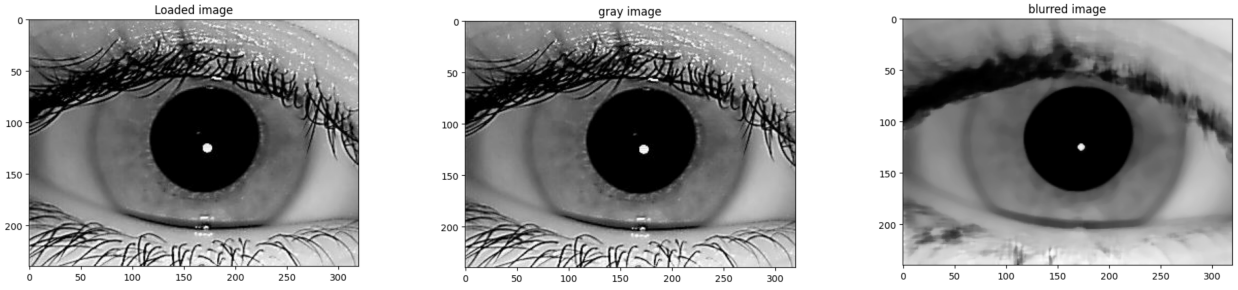
individuals. In parallel, the IIT Delhi dataset offered a more extensive repository, boasting 2,240 images collected from 224 individuals. This larger dataset enriched the diversity of the image pool, enabling a more robust examination of iris patterns and characteristics. Through curation and adherence to data integrity standards, both datasets were leveraged to ensure the representativeness and reliability of the collected image samples. The combined utilization of these datasets facilitated a exploration of iris recognition methodologies, laying the groundwork for subsequent preprocessing, model development, and system integration stages of the study.

3.4 Image Pre-processing

Firstly, the images were converted to grayscale. This conversion is essential because it simplifies the image to a single channel representing the intensity of each pixel, rather than retaining separate color channels (such as red, green, and blue). Converting the image to grayscale reduces the computational complexity of subsequent processing steps while preserving essential features necessary for iris segmentation and normalization. Additionally, grayscale images are less affected by variations in lighting conditions compared to color images, ensuring greater consistency and accuracy in subsequent processing stages.

In the initial stage of image processing, our experimentation involved comparing the effectiveness of two commonly used blur techniques: Gaussian blur and median blur. It was found that median blur proved to be more effective in smoothing the image while preserving edges, thereby facilitating easier edge detection. The rationale behind why blur makes edge detection easier lies in its ability to reduce noise and irregularities in the image, resulting in clearer and more defined edges, which are crucial for accurate segmentation.

Blurring was achieved using the built-in functions provided by OpenCV. These functions efficiently apply the chosen blur technique to the input image, enabling us to explore different blurring options and determine the most suitable approach for our specific preprocessing requirements. Refer to appendix Code Listing 1 for implementation.



(a) Main Image (b) Gray Image (c) Blurred Image

Figure 3.1: Preprocessing results

Iris segmentation involves isolating the iris region from the input eye image, a process that can be subdivided into several key steps. Initially, the segmentation process entails detecting various components within the eye image, including the pupil, iris, and eyelids. Following this, a series of steps are performed to achieve accurate segmentation:

3.5 Pupil Detection

In our specific application, the Circular Hough Transform to detect the pupils within the eye images was utilized. By providing a range of acceptable radii for the pupils, the algorithm was instructed to search for circular shapes within that radius range. OpenCV provides built-in functions for implementing the Circular Hough Transform, making it convenient and efficient to integrate into our image processing pipeline. This approach (Appendix:Code Listing 2) allowed us to accurately locate and extract the pupils from the input eye images, laying the groundwork for further analysis and segmentation of the iris region.

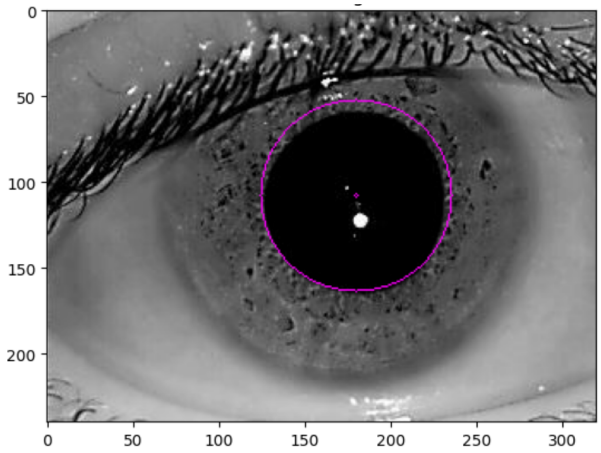


Figure 3.2: Pupil Detected from Image

3.6 Iris Detection

After successfully detecting the pupil, the next crucial step in iris detection involved identifying the edges of the iris. This process required the exploration of multiple methods to achieve accurate results. Initially, the Hough circle detection approach was employed; however, it proved ineffective as it struggled to properly discern the iris amidst the presence of eyelids and eyelashes, which caused disturbances in the search process. Subsequently, an attempt was made to differentiate along one dimension for intensity, selecting points with maximum intensity gradient to delineate the iris. Unfortunately, this method exhibited high susceptibility to noise, rendering it less reliable for iris detection. Finally, a novel approach was devised: the origin was transformed to the center of the pupil, and a radial intensity sum was calculated for radii ranging from 1.4 to 2.4 times greater than the pupil size, assum-

ing the iris fell within this size range. The point of maximum intensity sum difference was then estimated to correspond to the iris boundary. Notably, this method (Appendix: Code Listing 3) proved highly effective, particularly after applying an initial blur to the image, facilitating more precise and robust iris detection. If the iris radius falls below a certain threshold, it is regarded as a segmentation error, leading to the exclusion of such images from further processing. The following formulas were used:

$$I(r, \theta) = (y_c - r \sin \theta) \cdot \text{width} + (x_c + r \cos \theta) \quad (1)$$

$$R_{iris} = \arg\max\{I[r+2] + I[r+1] - I[r-1] - I[r-2]\} \quad (2)$$

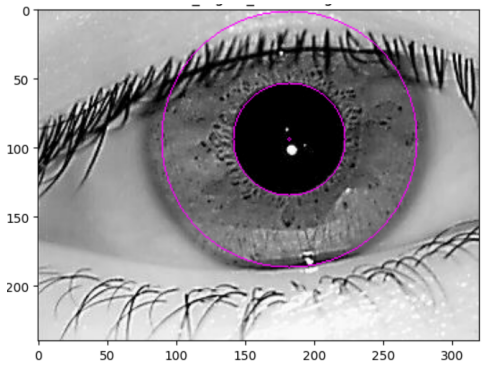


Figure 3.3: Iris Detected from Image

3.7 Eyelid Detection

Eyelids do not contribute distinctive patterns for iris recognition purposes and thus must be excluded from the analysis. To achieve this, our approach involves treating eyelids as straight lines. Initially, a search region extending from just above and just below the pupil with a padding of 5 pixels to account for noise, spanning the respective sides of the iris as shown in Figure 3.4 was defined.

Within this delineated region, Sobel filter in the horizontal direction to emphasize the edges indicative of eyelids was applied. This filtering process allows us to highlight potential eyelid boundaries. Subsequently, for each row within the designated region, we calculate the Sobel magnitude, representing the intensity of edge gradients. These magnitudes are then normalized to ensure consistency across rows. The next step involves identifying the row with the greatest normalized Sobel magnitude, indicative of the presence of an eyelid. Should this magnitude exceed a predetermined threshold, which was found to be effectively set at 0.05 for our purposes, the corresponding row is deemed to represent the eyelid. By

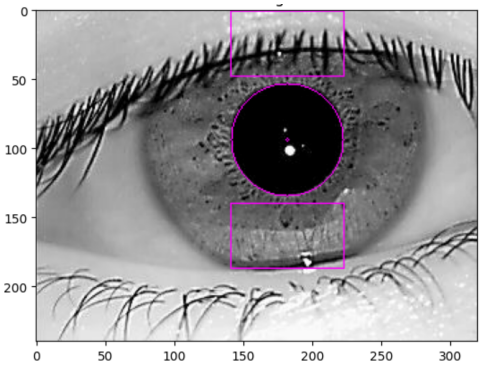


Figure 3.4: Search Region for Eyelids

employing this methodology, the accuracy and robustness of the iris segmentation process were enhacned.

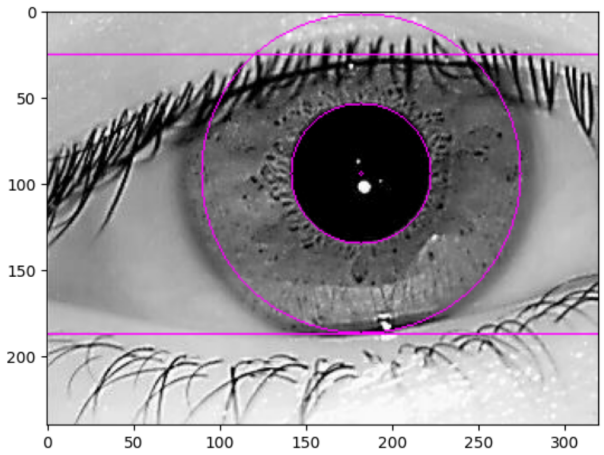


Figure 3.5: Eyelids segmented

3.8 Masking

Incorporating the derived parameters such as the pupil radius, center coordinates, and iris radius, a mask was constructed. This mask (Appendix: Code Listing 7) serves as a binary template, delineating the region corresponding to the iris within the input image. The creation of this mask involves defining a circular area centered at the pupil coordinates, with a radius equal to the estimated iris radius. Subsequently, bitwise AND operation between the generated mask and the original image was performed. This operation effectively retains only the pixels within the iris region, while discarding the surrounding areas. By applying this masking technique, isolation and extraction the iris from the input image was done, facilitating further analysis and feature extraction processes. An example of mask is shown in Figure 3.6

Then using simple calculations from eyelid rows, the eyelid pixels were removed from the

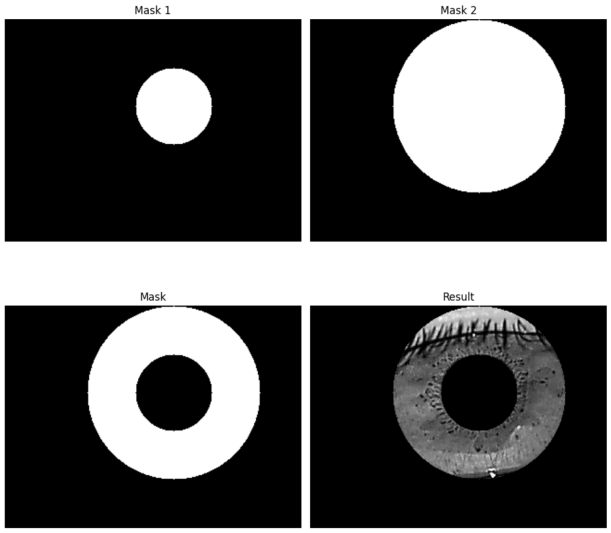


Figure 3.6: Masking

masked image.

3.9 Image Normalization

After masking the iris region, the subsequent step involves image normalization. This process begins with rubber sheet mapping, where the masked image is transformed into polar coordinates. A custom rubbersheet mapping function was used (see Appendix:Code Listing 8 and **??**).

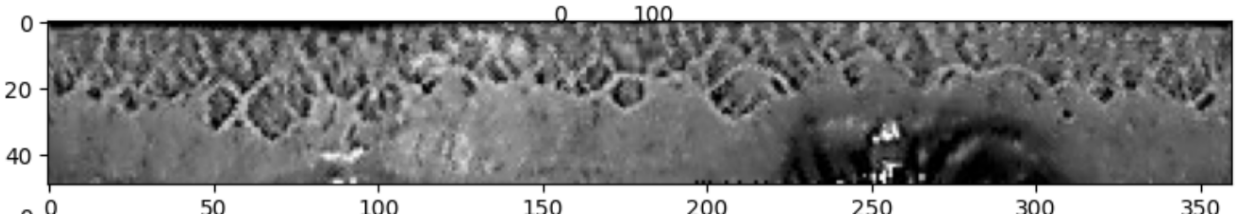


Figure 3.7: Normalized Iris

However, despite the transformation, the normalized iris image often exhibits low contrast and non-uniform illumination,as we can see a normalized image in Figure 3.7, attributed to variations in the light source position during image capture. To address these issues, image enhancement techniques are employed to compensate for the lack of contrast and illumination inconsistencies.

One such technique involves local histogram analysis applied to the normalized iris image. This method aims to mitigate the impact of non-uniform illumination by locally adjusting the intensity levels of the image. By analyzing small regions of the iris image individually, local histogram analysis ensures that each portion of the iris receives appropriate adjustments,

leading to a well-distributed texture image. This enhanced image, as in Figure 3.8 provides a more accurate representation of the iris features, thereby improving the performance of subsequent iris recognition algorithms. Through the combination of rubber sheet mapping and local histogram analysis, image normalization enhances the quality and consistency of iris images.

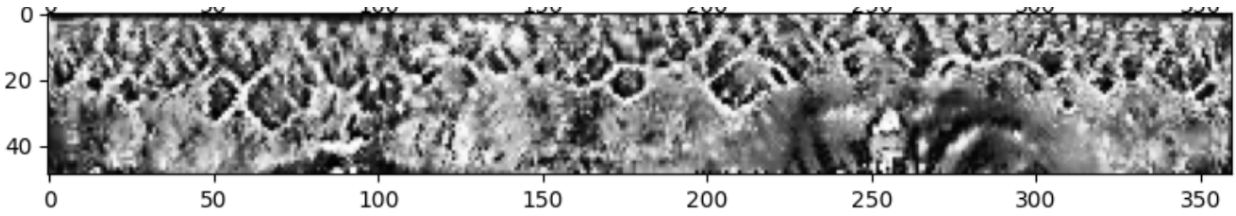


Figure 3.8: Enhanced Iris

3.9.1 Batch Normalization

Batch normalization technique to normalize all images in a batch, ensuring consistent scaling and centering across the dataset were used. These normalized images were then organized into separate folders, ready to be fed into the neural network for training. By preprocessing the entire batch of images in this manner, effectiveness of the labeled data and streamline the training process was increased. These normalized data were used for training algorithm further down.

3.10 Data Augmentation

To increase the number of data samples and to introduce a diversity of data, augmentation techniques were applied (Fig:3.9 to Fig:3.13). Before augmentation, only about $9,548$ pairs could be generated but after augmentation the number of pairs increased to $21,172$.

These techniques were applied in random order to randomly selected images ensuring that no two augmented images are same.

3.11 Data Pair Generation

Data pair generation was performed for our Siamese neural network using the normalized iris images. Positive pairs consists of normalized iris images belonging to the same individual,

Figure 3.9: Application of gaussian blur in normalized image

Figure 3.10: Addition of noise in normalized image

Figure 3.11: Contrast Variation

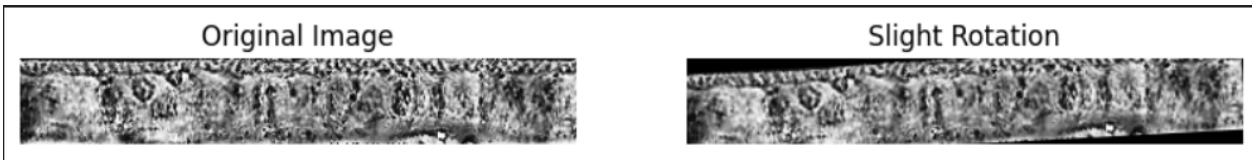


Figure 3.12: 5 degree rotation

Figure 3.13: 10 units of translation

```
1   image1,image2
2   iit_101_08_R.bmp,iit_198_04_L.bmp
3   iit_115_10_R.bmp,iit_150_06_R.bmp
4   iit_104_08_R.bmp,iit_091_01_L.bmp
5   iit_003_06_L.bmp,iit_062_04_L.bmp
```

Figure 3.14: Snapshot of generated csv

ensuring they are from either the left or right eye. Negative pairs are formed by combining iris images from different individuals, irrespective of their laterality, or by pairing the left and right iris images of the same individual.

The pairs thus generated were saved in csv files with two columns image1 and image2. Then, these csv files were read using `read_csv` a `Pandas` function into data frames `pos_df` and `neg_df` and label column (0s for `neg_df` and 1s for `pos_df`) were added to both of the dataframes. They were combined and shuffled into a single dataframe.

3.12 Data Pipelining

In this section, we describe the construction of the data pipeline using TensorFlow's `tensorflow.data` API. The data pipeline serves as a crucial component in efficiently preparing and feeding the dataset into the model for training. A data pipeline refers to a series of data processing steps that transform raw input data into a format suitable for training a machine learning model. In the context of our project, the data pipeline involves loading image data, preprocessing it, and batching it to create mini-batches for training.

3.12.1 Utilizing tf.data.Dataset

We use `tensorflow.data.Dataset` abstraction of `tf.data` API that represents a sequence of elements, in which each element consists of two images(positive or negative pair) and a label. In our project, we create it from a data source(data frame) using `from_tensor_slices` method of `tensorflow.data` API.

3.12.2 Mapping

In the context of our project, mapping refers to the process of applying a transformation function to each element of the `tf.data.Dataset`. We used the `map` method to apply a preprocessing function `load_and_preprocess_image` to each image enabling parallel processing with the `num_parallel_calls` parameter.

3.12.3 Batching

After mapping the, we batch the datasets into mini-batches of a given size (e.g. 32) using the `batch` method. Batching offers computational efficiency through parallel processing, memory efficiency by reducing overhead, and stability in training by providing more stable gradient estimates, thus enhancing convergence during optimization.

3.12.4 Prefetching

We also use the `prefetch` method of tensorflow.data API to allow data pipeline to overlap the preprocessing of data with the training of the model. When prefetching is applied to a dataset using the `prefetch` method, TensorFlow creates a background thread to fetch batches of data ahead of time, while the current batch is being processed by the model. The `tensorflow.data.experimental.AUTOTUNE` parameter is used with the prefetch method which allows TensorFlow to dynamically adjust the prefetching buffer size based on available resources and system load, further enhancing efficiency.

3.13 Siamese Network Implementation

The Siamese Network was implemented in sections elaborated as follows:

3.13.1 Transfer Learning

Transfer learning involves reusing a pre-trained model developed for one task to bootstrap the learning process for a related task, thereby accelerating training and improving performance with a smaller dataset. In our project, transfer learning is utilized in two specific areas:

1. Feature Extraction: The base convolutional network, such as **VGG16** with pretrained **ImageNet** weights, already contains generic features useful for image classification. We use this base network by fetching it while passing parameter `trainable = False` and `include_top = False` for the base model, its parameters are frozen, allowing us to extract and utilize these generic features while adapting only the task-specific classification layers to the new dataset. This means that the weights, biases and other parameters are kept fixed which was learnt from ImageNet and only the layers except them undergo changes in parameters.

2. Fine-Tuning: While finetuning a model for better performance, we unfreeze a few of the top layers of a frozen model base and jointly train added layers(for our case distance and Dense layers) and the last layers of the base model. Now, the unfrozen convolutional layers undergo changes in weights. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more

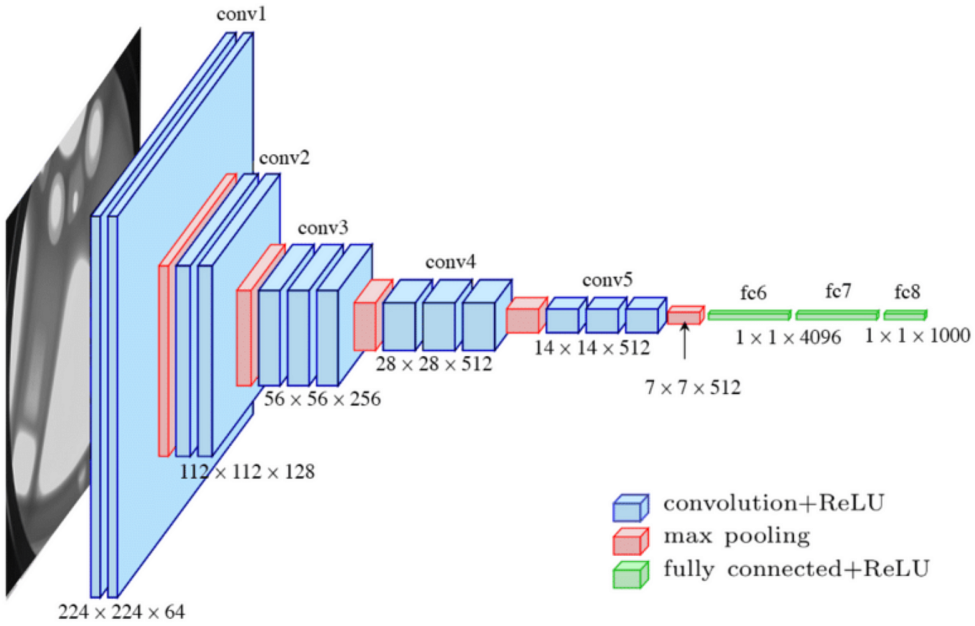


Figure 3.15: VGG16 architecture

relevant for the specific task.

3.13.2 VGG

The VGG architectures were introduced as part of the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Karen Simonyan and Andrew Zisserman in 2014. The primary motivation behind VGG was to explore the impact of network depth on the performance of CNNs. One of the key aspects of VGG networks is their uniformity in design: they predominantly use 3x3 convolutional filters with a stride of 1 and 2x2 max-pooling windows with a stride of 2. This uniform structure allows for easy implementation and understanding, making VGG networks a popular choice for both research and practical applications.

We use the following variants of VGG in our project.

VGG16

The VGG16 was one of the six configuration suggested by the Visual Geometry Group. The VGG16 imported from keras.applications has 16 weight layers with 138 million trainable parameters.

3.13.3 ResNet

ResNet, short for Residual Network, is a deep convolutional neural network architecture proposed by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun from Microsoft

Research in their paper "Deep Residual Learning for Image Recognition" in 2015. ResNet was designed to address the vanishing gradient problem, which arises when training very deep neural networks. ResNet introduces the concept of residual learning, where the network learns the residual mapping instead of the full mapping.

1. Full Mapping: If we denote the desired underlying mapping as $H(x)$, where $H(x)$ represents the input to a particular layer, then the traditional approach aims to learn $H(x)$ directly. In mathematical terms, the output of a layer can be denoted as $F(x)$, where $F(x) = H(x)$.

2. Residual Mapping: Instead of learning $H(x)$ directly, ResNet learns the residual mapping $F(x) - x$. The output of a layer is obtained by adding this residual mapping to the input, resulting in $H(x) = F(x) + x$.

ResNet50

ResNet50 is a variant of ResNet architecture which consists of 50 layers, including convolutional layers, residual blocks, and fully connected layers. It surpasses several architectures like VGG, AlexNet, Inception in terms of performance in image recognition tasks. We utilize this variant of ResNet architecture in our project.

3.13.4 Dimensionality Reduction and Feature Aggregation

The base convolutional model's output is passed to a `GlobalAveragePooling2D` layer. This layer reduces spatial dimensions of input feature maps while retaining channel depth by averaging values across height and width. This aggregation captures important features across feature maps, aiding in spatial generalization and dimensionality reduction for compatibility with subsequent layers.

3.13.5 Distance calculation

Now, the two parallel outputs from the `GlobalAveragePooling2D` layer is passed to a layer which computes the absolute difference between pooled feature representations of two input normalized iris images. This layer outputs how far apart or close are the features of the two images from each other. This layer is implemented using a custom `lambda` function which takes a single argument `x` with two input tensors, subtracts them and outputs the absolute value using `K.abs`.

3.13.6 Similarity Score

The final layer of the Siamese Network is a `Dense` layer with 1 output and `sigmoid` activation which takes input the distance vector and outputs a value from 0 to 1 (similarity score)

depending on how dissimilar or similar the images are respectively.

3.13.7 Loss function

Since the output of Siamese Network is binary in nature (0 or 1), we primarily used `Binary Cross-Entropy` as our loss function. Also, called as log loss it measures the dissimilarity between the true labels and the predicted labels for each sample in the dataset.

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{3.1}$$

Where,

- $L(y, \hat{y})$ is the binary cross-entropy loss.
- N is the number of samples.
- y_i is the true label (0 or 1) for sample i.
- $\hat{y}$ is the predicted label for sample i.

However, we also experimented with `Contrastive loss` but to our misfortune, it did not work as expected and resulted in Siamese Network unable to learn.

$$L(y, d) = \begin{cases} \frac{1}{2} d^2 & \text{if } y = 1 \text{ (similar pair)} \\ \frac{1}{2} \max(margin - d, 0)^2 & \text{if } y = 0 \text{ (dissimilar pair)} \end{cases} \tag{3.2}$$

Where,

- $L(y, d)$ represents the contrastive loss.
- y represents the true label (0 for negative pair and 1 for positive pair).
- d represents distance between images (calcualted in the distance layer).
- $margin$ is a hyperparameter that defines minimum distance threshold between dissimilar pairs.

3.13.8 Optimizer

An optimizer is used in machine learning to facilitate the process of Gradient Descent and loss minimization as efficiently and quickly as possible. In our project, we have used a widely used optimizer, **Adaptive Moment Estimation (Adam)** optimizer which integrates the features from `Adaptive Gradient Algorithm (AdaGrad)` and `Root Mean Square Propagation (RMSProp)`.

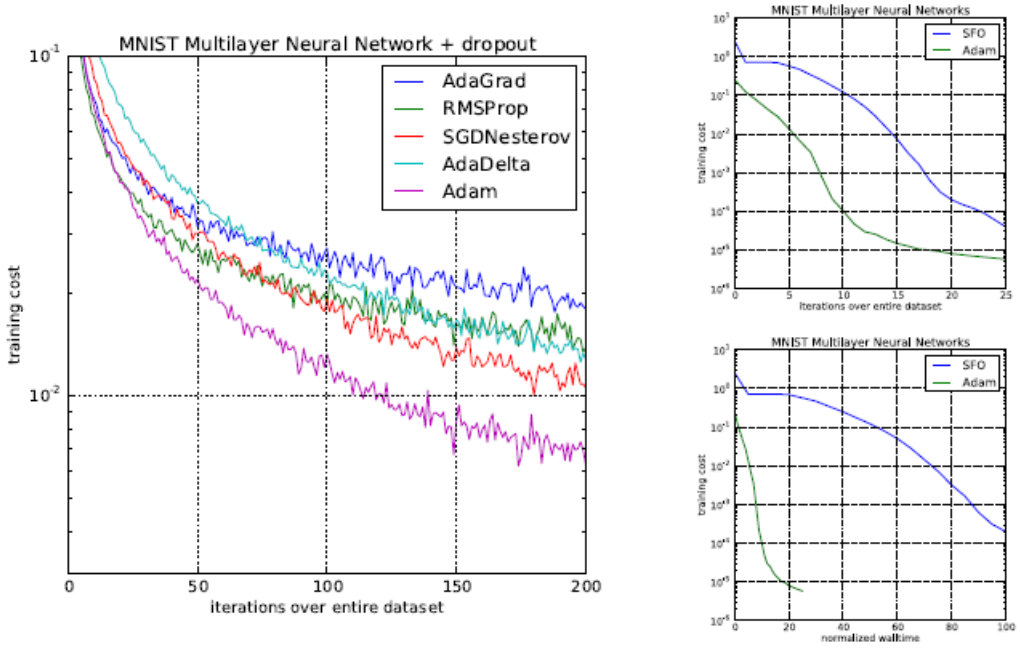


Figure 3.16: Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron Source: A Method for Stochastic Optimization, 2015.

3.13.9 Validation and Testing

The trained model undergoes k-fold cross validation using the cross-validation data (obtained from splitting before training). Then, evaluation is performed by observing model's score on the parameters like accuracy, sensitivity and specifity and required changes are performed to combat overfitting or underfitting.

Finally, the model undergoes testing using the test data and it's performance on the various parameters are noted down.

3.14 Implementation

The implementation process began with a recognition of the essential practicality of a model: its usability in real-world scenarios. With this in mind, the team set out to integrate the iris recognition model into a user-friendly website, facilitating secure user authentication and document storage.

3.14.1 Website Development

The development of the web application involved a concerted effort to employ modern web development technologies. HTML, CSS, and JavaScript were utilized for the frontend to ensure a responsive and intuitive user interface. Meanwhile, Python, coupled with Django framework, was chosen for the backend to manage user authentication, database operations,

and interactions with the iris recognition model.

3.14.2 User Authentication

Central to the implementation was the establishment of a robust user authentication system to safeguard user data. A secure authentication mechanism was implemented allowing users to sign up using their iris images, securely stored within the database. During login, users will be prompted to authenticate using their iris image, username, and password for verification.

3.14.3 Iris Recognition Integration

The core functionality of the implementation relied on seamlessly integrating the iris recognition model into the web application. Upon login, users were prompted to capture their iris image using their device's camera. The captured image underwent processing by the iris recognition model to authenticate the user's identity. Successful authentication granted access to the user's account for document management.

3.14.4 Document Storage

Beyond authentication, the implementation included functionality for document storage and access. Users could upload documents to their account, which were encrypted and securely stored in the database. Accessing documents required re-authentication using the user's iris image, ensuring that only authorized users could view sensitive information.

3.14.5 User Experience

Throughout the implementation, the team prioritized delivering a seamless and intuitive user experience. The website's design aimed for user-friendliness, providing clear instructions for iris image slection and document management. Furthermore, performance optimization efforts were undertaken to ensure smooth user experiences across various devices and network conditions.

The goal was to enable real-time iris image capture; however, the lack of resolution in webcams made this impractical. As a result, the current version of the website allows users to choose images from their computers. Nevertheless, integrating real-time image capture with appropriate hardware is straightforward and requires only minor adjustments to the website code, while preserving the existing architecture.

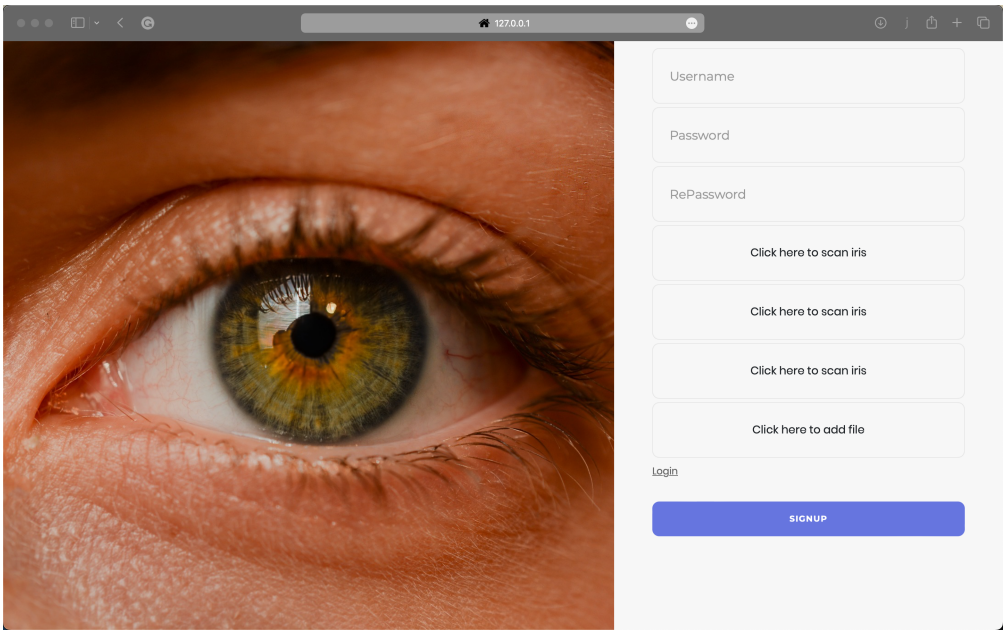


Figure 3.17: Signup page

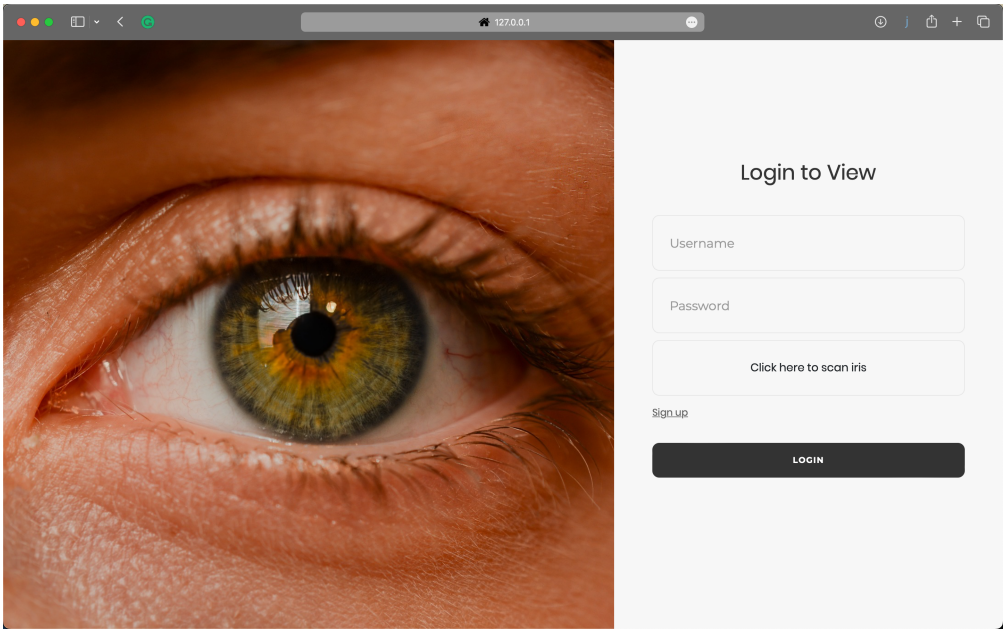


Figure 3.18: Login page

4. Experimental Setup

4.1 Data Collection

As mentioned earlier, we used IITD and MMU data sets which contain high resolution images of eyes of people from various age group. To increase the number of available samples, data augmentation techniques like crop, brightness shift, gamma shift, rotation and contrast shifts were applied which taking our trainable samples to above 10000.

4.2 Software Specification

The experimental setup was conducted using the following software specifications:

- **Python Version:** 3.10.12
- **Libraries Used:**
  - OpenCV: version 4.9.0.8
  - NumPy: version 1.2.64
  - TensorFlow: version 2.15.0
  - Scikit-learn: version 1.2.2
  - Matplotlib: version 3.8.3
  - jupyter-client: version 8.6.0

4.3 Image Normalization Setup

The experimental setup for image normalization involved conducting the process within a Jupyter Notebook environment. The initial steps included loading the dataset containing the images to be normalized and preprocessing them to ensure uniform dimensions and color channels across all images. Subsequently, the image normalization algorithm was implemented using Python code, leveraging libraries such as OpenCV, NumPy, and Matplotlib. Within the Jupyter Notebook, individual code blocks were executed to apply the normalization technique to the images iteratively. Following the normalization process, the normalized images were analyzed to assess the effectiveness of the normalization procedure.

The individual code blocks executed in the Jupyter Notebook for image normalization can be found in the appendix section of this document.

The images were organized into folders. Subsequently, Python scripts were developed to systematically access each image, assign a unique identifier, normalize it, and then store it in a designated folder. This process ensured that the images were appropriately prepared for downstream tasks, such as training neural networks or conducting further analysis within the pipeline.

4.4 Training Setup

The training process commenced with the parsed normalized images obtained from the preprocessing stage. Thus obtained images were passed to augmentation script to add blur and random transformations to introduce variablility and increase the number of data. These images were meticulously organized and processed using a custom Python script specifically designed for this task. The script meticulously traversed through the normalized image dataset, assigning unique identifiers to each image and subsequently generating CSV files containing pairs of positive and negative examples. This step was crucial for establishing the training data's structure and facilitating efficient model training.

Following the creation of the CSV files, TensorFlow's input-output functionalities were leveraged to load the data pairs into memory. Each pair was meticulously labeled according to its classification, setting the stage for supervised learning within the model training process. Additionally, to ensure robust validation and evaluation of the model's performance, the dataset was divided into multiple folds using a k-fold approach. This partitioning strategy enabled the model to be trained and evaluated across different subsets of the dataset, enhancing its generalizability and robustness.

With the dataset appropriately prepared and partitioned, the next step involved constructing and compiling the model architecture. This process entailed defining the neural network's layers, activation functions, loss function, optimizer, and evaluation metrics. The model architecture was meticulously designed to accommodate the complexities of the iris recognition task and optimize its performance across various metrics.

Once the model was constructed and compiled, the training process commenced. The input data, organized into batches, were fed into the model for multiple epochs. During each epoch, the model iteratively learned from the training data, adjusting its parameters to minimize the defined loss function and improve its performance. The training process continued until a predefined number of epochs were completed, or until convergence criteria were met.

Then the model underwent finetuning where higher convolutional layers were unfreezed to set parameters specific to iris recognition.

Dataset Partitioning

The dataset used for performance evaluation was carefully partitioned into distinct subsets, including training, validation, and test sets. The splits yielding good results was 70-20-10 split resulting in best validation and test accuracy.

Evaluation Metrics

Several evaluation metrics were employed to quantitatively assess the model's performance. These metrics included accuracy, loss, precision, , sensitivity, specificity and F1 score.

Cross-Validation

To ensure robustness and mitigate the effects of dataset bias, cross-validation techniques were applied during performance evaluation. Specifically, k-fold cross-validation was employed, where the dataset was divided into k subsets (folds), and the model was trained and evaluated k times, with each fold serving as the validation set once and the remaining data as the training set.

Visualization

In addition to numerical metrics, visualization techniques were utilized to gain a deeper understanding of the model's performance. This included generating confusion matrices, loss curves, and precision-recall curves to visualize the model's classification performance and identify any potential areas for improvement.

5. System design

5.1 Use Case Diagram

The use case diagram illustrates the interactions between actors and the system functionalities.

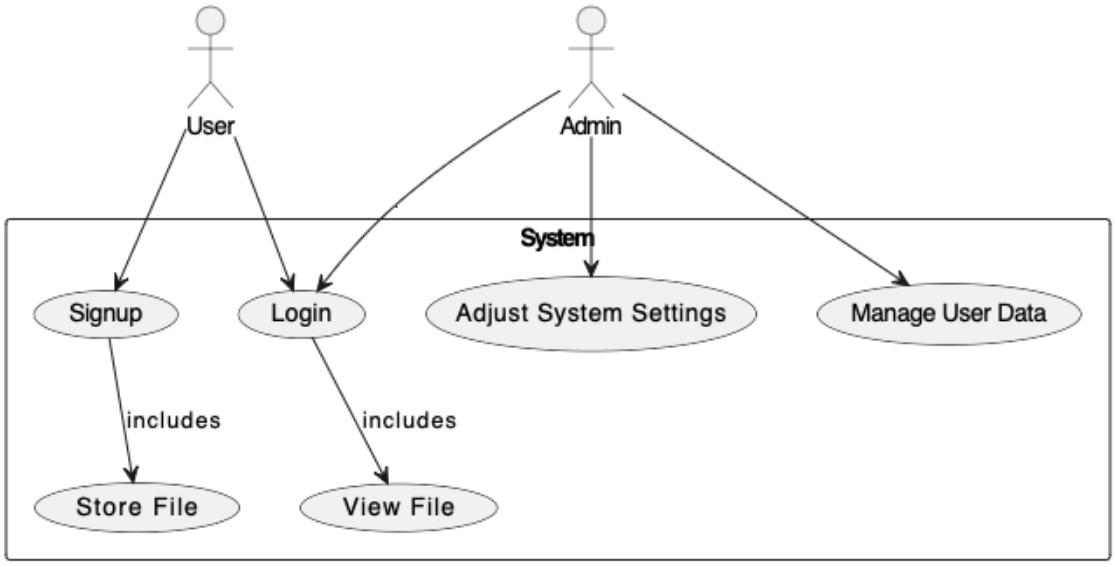


Figure 5.1: Use Case Diagram

5.2 System Sequence Diagrams

5.2.1 Login Use Case

The system sequence diagram for the login use case illustrates the sequence of interactions between the user interface, preprocessor, recognition model, and database during the login process. As shown in Figure 5.2, the user enters their credentials and iris image, which are then processed and verified by the system. If the verification is successful, the user is authenticated and granted access to the system. Otherwise, the login attempt is unsuccessful, and the user is notified accordingly.

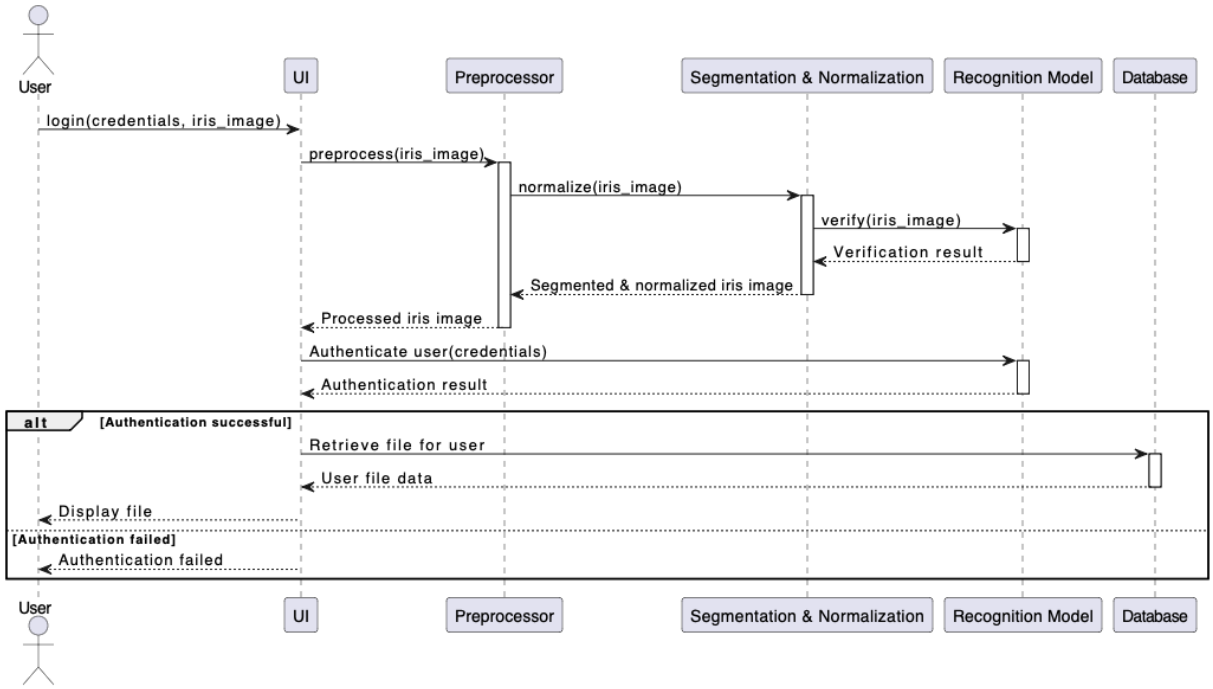


Figure 5.2: System Sequence Diagram for Login Use Case

5.2.2 Signup Use Case

The system sequence diagram for the signup use case depicts the sequence of interactions between the user interface, preprocessor, recognition model, and database during the signup process. As depicted in Figure 5.3, the user provides their credentials, iris images, and a file for safekeeping. The system preprocesses and verifies the iris images, and if successful, stores the user data and file in the database. The user is then notified of the successful signup. If the verification fails, the signup attempt is unsuccessful, and the user is notified accordingly.

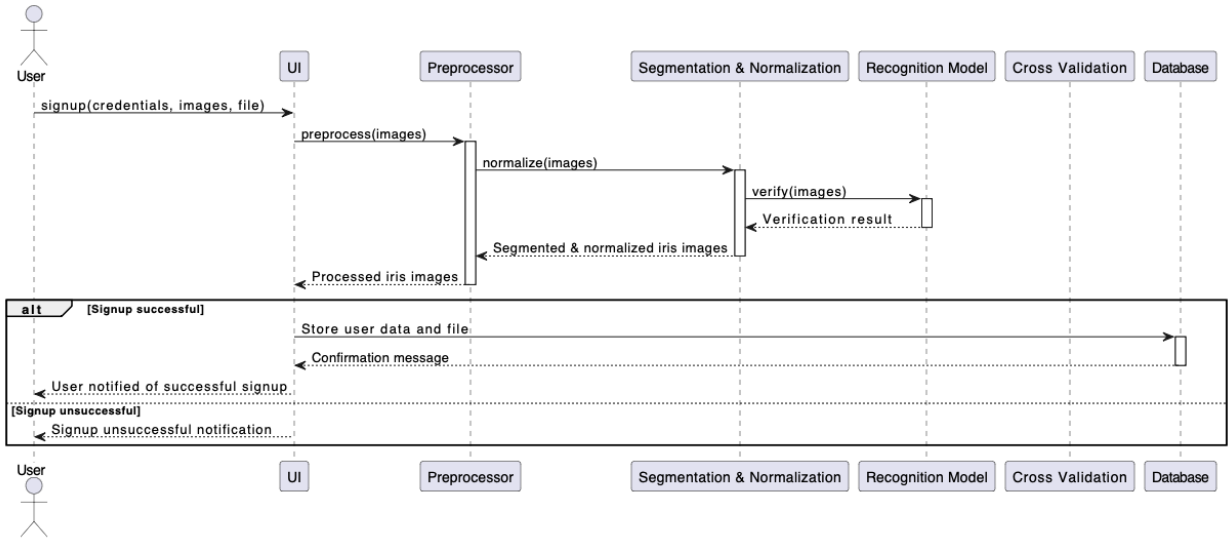


Figure 5.3: System Sequence Diagram for Signup Use Case

5.3 System Block Diagram Overview

The system block diagram illustrates components and their functions within an iris recognition system:

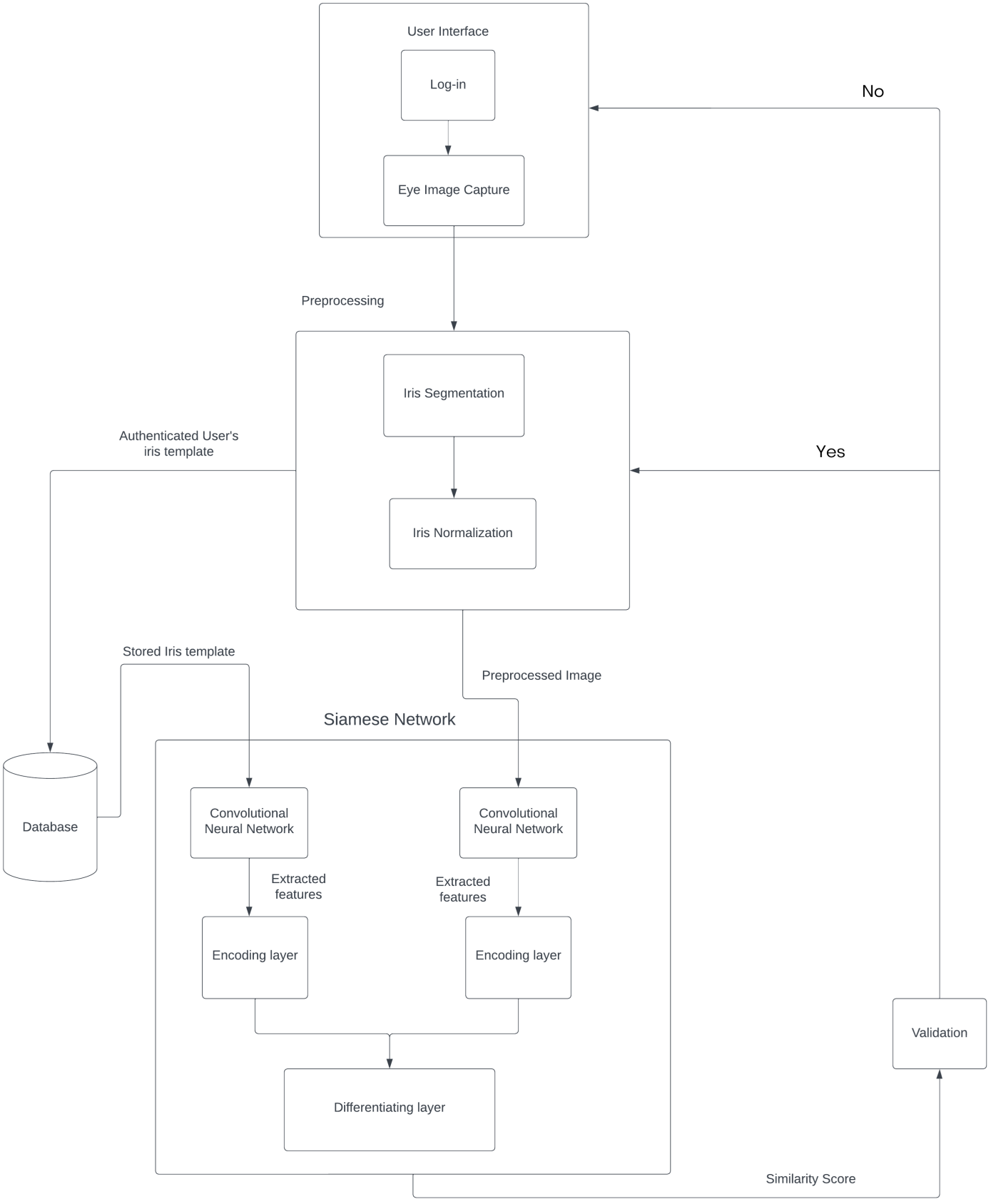


Figure 5.4: System Block Diagram

6. Results and Discussions

6.1 Training and Validation

The training and validation split was $80 - 20$ and batch size used was 32.

6.1.1 VGG16

Training accuracy of 85.44% and validation accuracy of 84.21% was obtained. Similarly, training and validation loss were 0.4098 and 0.4078 fig:6.1.
After finetuning for 5 epochs, significant increase in both training and validation accuracy and significant decrease in both training and validation loss were observed fig:6.2. The training and validation accuracy were 97.38% and 95.23%. Similarly, training and validation loss were 0.1771 and 0.1849.

Cross Validation

Five-fold cross-validation was conducted on the VGG16 model to assess the robustness of its performance across different fold configurations. The plot depicted in Fig: 6.3 illustrates that the model's accuracy and loss values remained consistent across all folds, indicating the stability of its predictive capabilities.

6.1.2 ResNet50

ResNet50 was also fed same dataset. The training and validation accuracy thus obtained was 88.63% and 87.26% respectively. The observed training and validation loss and accuracy plot is shown fig: 6.4.
After finetuning, significant increase in both training and validation accuracy and significant decrease in both training and validation loss were observed fig:6.5. The training and validation accuracy were 97.38% and 95.23%. Similarly, training and validation loss were 0.1771 and 0.1849.

6.2 Testing and Evaluation

The models were tested on test data which predominantly consisted of MMU data (the data model has not seen at all) and very few (IITD data). Testing on MMU data only resulted on less accuracy due to the fact that MMU data's resolution is poor compared to IITD data. ROC curve and Confusion matrix were obtained which were used for calculating performance

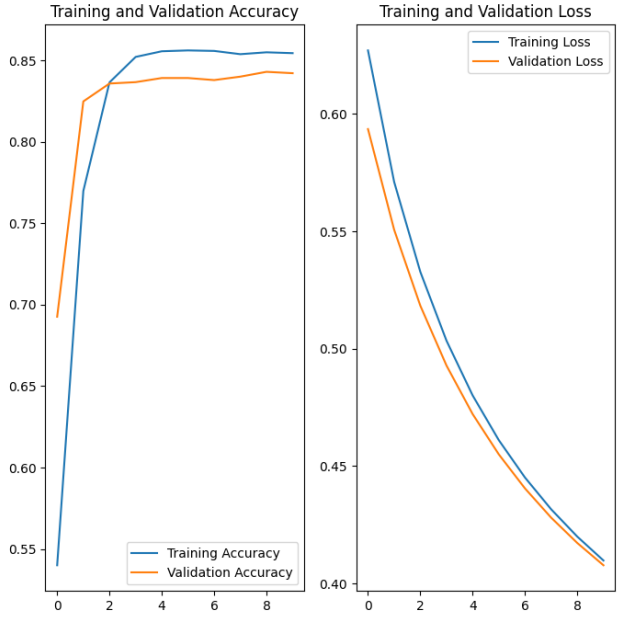


Figure 6.1: Training and Validation loss and accuracy plot of VGG16

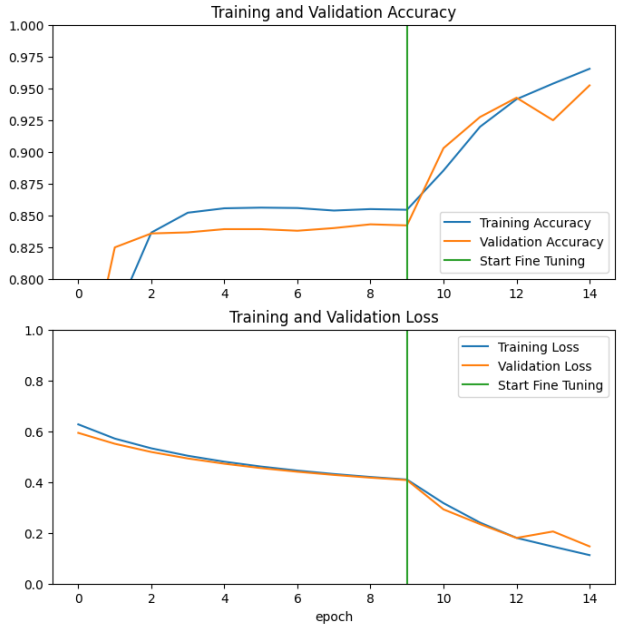


Figure 6.2: Training and Validation loss and accuracy plot of VGG16 including finetuning

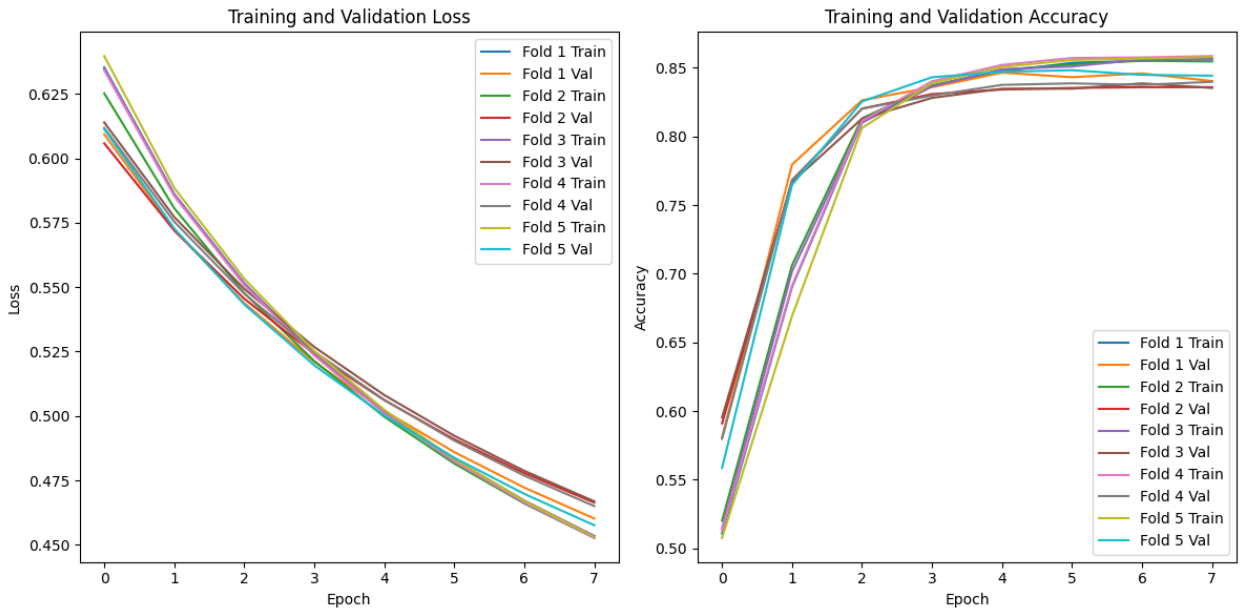


Figure 6.3: 5 fold cross validation on VGG16

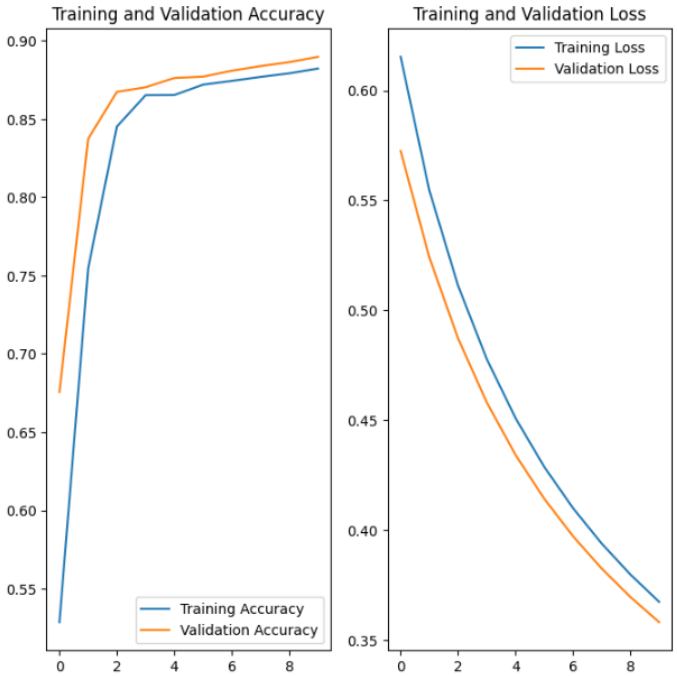


Figure 6.4: Training and Validation loss and accuracy plot of ResNet50

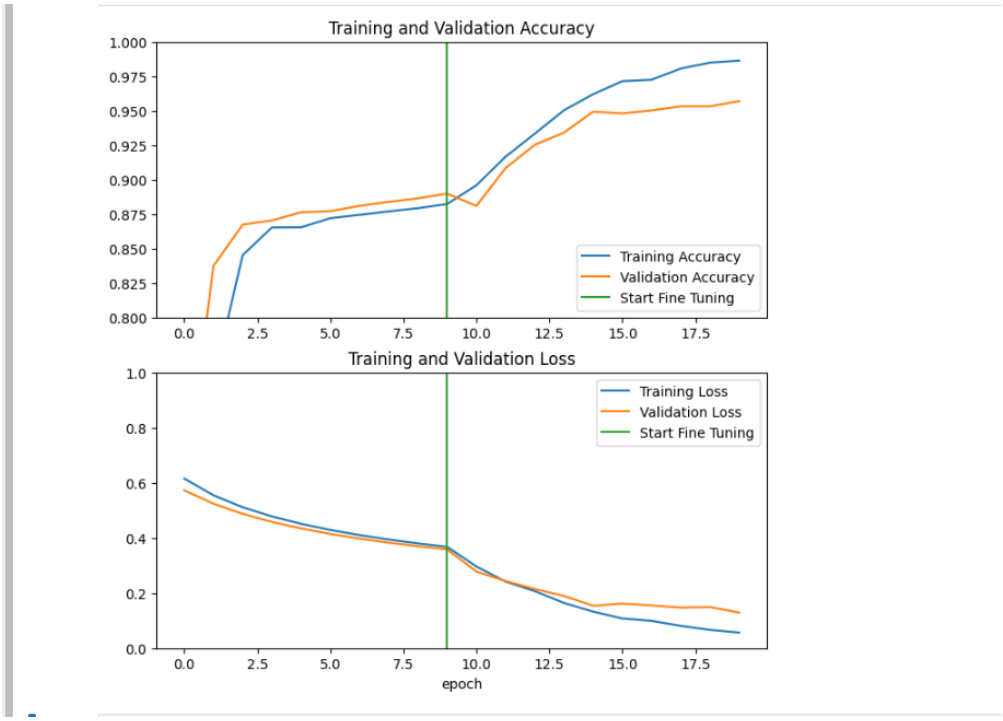


Figure 6.5: Training and Validation loss and accuracy plot of ResNet50 including finetuning

metrics.

ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a model that outputs a binary score across different threshold settings. It plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. A confusion matrix presents a summary of the predictions made by the model against the actual labels or ground truth across different classes.

- True Positive (TP): The number of instances that are correctly predicted as positive by the model.
- True Negative (TN): The number of instances that are correctly predicted as negative by the model.
- False Positive (FP): Also known as Type I error, it represents the number of instances that are incorrectly predicted as positive by the model when they are actually negative.
- False Negative (FN): Also known as Type II error, it represents the number of instances that are incorrectly predicted as negative by the model when they are actually positive.

6.2.1 VGG16

The Area Under the Curve (AUC) of ROC curve for VGG16 Fig:6.6 thus obtained is 0.76 which is a satisfactory result considering the fact that the test dataset (MMU dataset) had

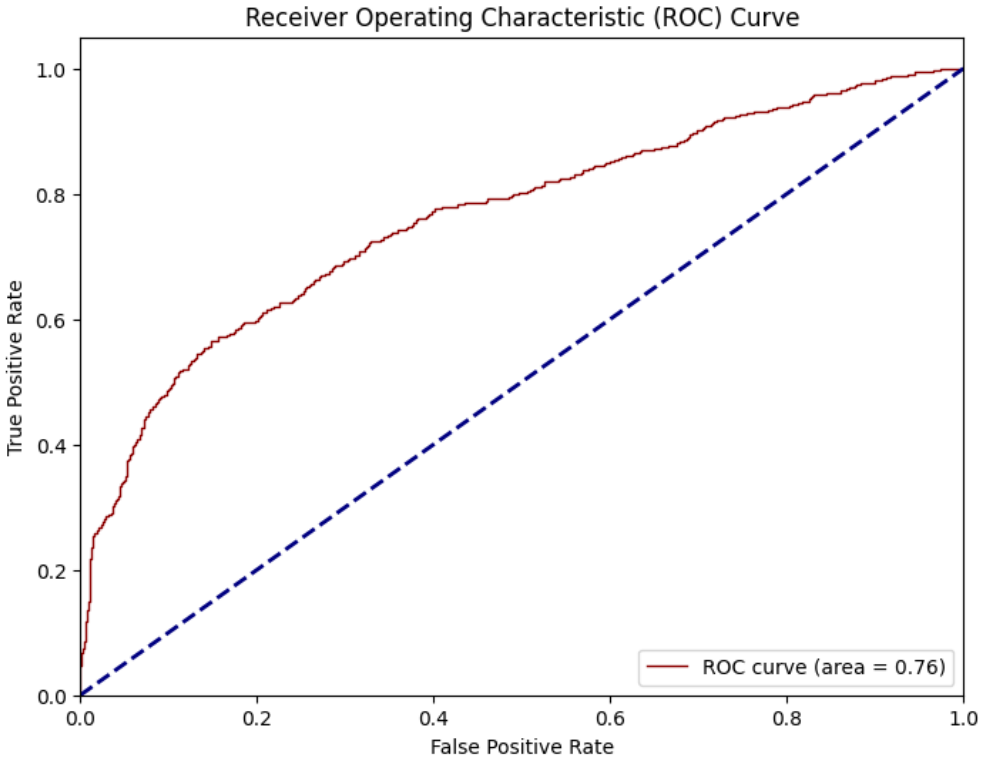


Figure 6.6: ROC curve of VGG16 in MMU test dataset

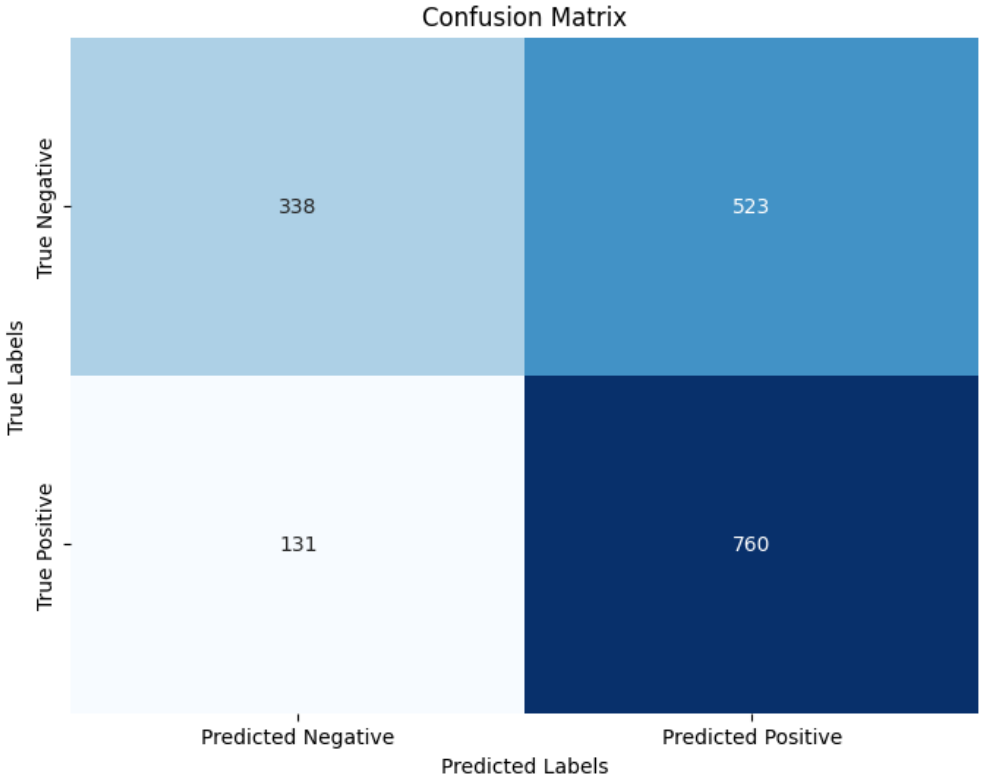


Figure 6.7: Confusion matrix of VGG16 in MMU test dataset

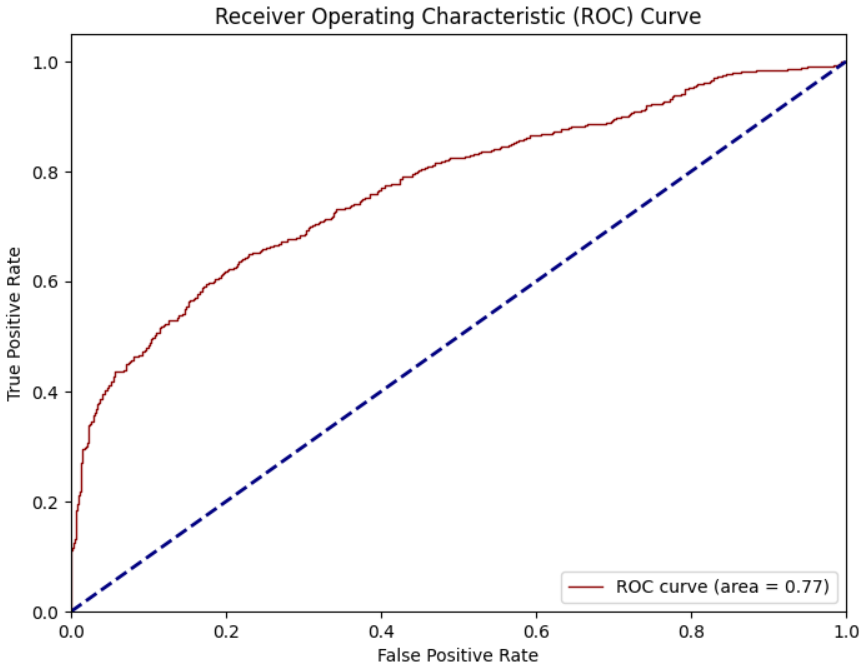


Figure 6.8: ROC Curve of ResNet50 on MMU dataset

a poor resolution compared to IITD dataset. From the confusion matrix of VGG16 Fig:6.7, following values were obtained:

- TP = 760
- TN = 338
- FP = 523
- FN = 131

$$Precision = \frac{TP}{TP + FP} = 0.5923 = 59.23\% \tag{6.1}$$

$$Recall(sensitivity) = \frac{TP}{TP + FN} = 0.8529 \tag{6.2}$$

$$Specificity = \frac{TN}{TN + FP} = 0.3925 \tag{6.3}$$

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} = 0.6991 \tag{6.4}$$

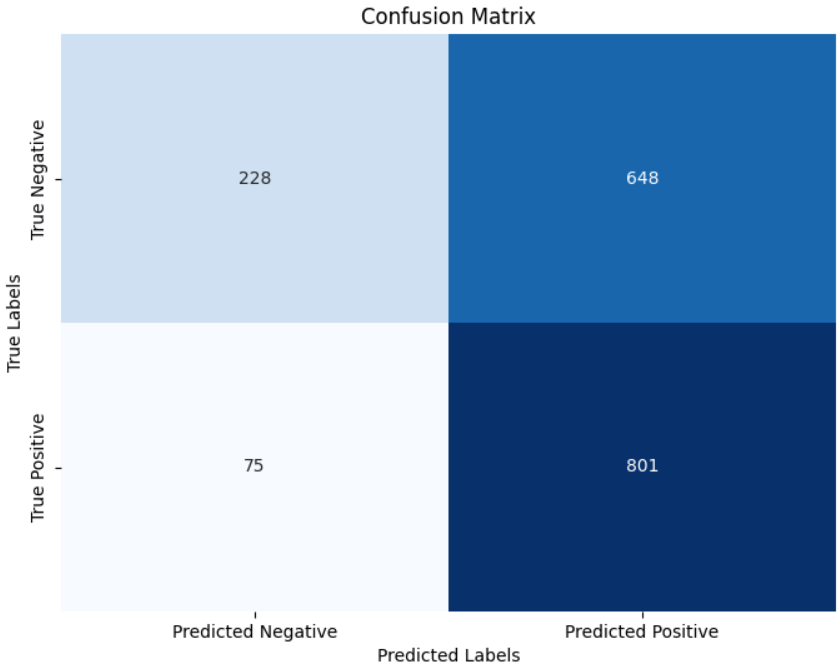


Figure 6.9: Confusion matrix of ResNet50

6.2.2 ResNet50

From confusion matrix of ResNet, the following values are obtained:

- TP = 801
- TN = 228
- FP = 648
- FN = 75

$$Precision = \frac{TP}{TP + FP} = 0.5528 = 55.28\% \tag{6.5}$$

$$Recall(sensitivity) = \frac{TP}{TP + FN} = 0.9143 \tag{6.6}$$

$$Specificity = \frac{TN}{TN + FP} = 0.2602 \tag{6.7}$$

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} = 0.689 \tag{6.8}$$

7. Limitations and Future Enhancements

7.1 Limitations

Despite its capabilities, the iris recognition system presented in this study has several limitations that should be considered:

7.1.1 Hardware Limitations

One of the main limitations of the system is its dependence on specialized hardware for iris image capture. This requirement limits the accessibility of the system, especially in resource-constrained environments where such hardware may not be readily available or affordable.

7.1.2 Environmental Constraints

The performance of the iris recognition system may be affected by environmental factors such as lighting conditions and occlusions. Variations in lighting can impact the quality of iris images captured, leading to reduced accuracy in recognition. Additionally, occlusions such as glasses or contact lenses may obstruct the view of the iris, further hindering the system's performance.

7.1.3 User Cooperation

The effectiveness of the system relies on the cooperation of users during the enrollment and authentication processes. Users are required to position themselves correctly in front of the camera for iris image capture and ensure that their eyes are fully visible. Failure to comply with these requirements may result in inaccurate recognition or rejection by the system.

7.1.4 Security Concerns

While iris recognition is considered a secure biometric modality, it is not immune to security threats such as spoofing or presentation attacks. Adversaries may attempt to deceive the system by presenting counterfeit iris images or using sophisticated techniques to bypass authentication. Ensuring robust countermeasures against such attacks is essential to maintain the system's integrity and reliability.

7.1.5 Scalability

The scalability of the system may be limited by factors such as processing power and database size. As the number of users and iris templates stored in the database increases, the com-

putational complexity of matching and verification processes may also escalate, potentially leading to performance degradation and slower response times.

7.1.6 Ethical and Privacy Considerations

Deploying an iris recognition system raises ethical and privacy concerns regarding the collection, storage, and use of biometric data. It is essential to implement stringent data protection measures and comply with relevant privacy regulations to safeguard users' rights and prevent unauthorized access or misuse of sensitive information.

7.2 Future Enchancements

While the current iteration of the iris recognition system has demonstrated promising capabilities, there are several areas for future enhancement and refinement:

7.2.1 Hardware Integration

To address the limitations posed by low-resolution laptop webcams, future enhancements may involve the integration of specialized hardware components with higher image quality and near-infrared (NIR) capabilities. This upgrade would enhance the system's ability to capture clear and accurate iris images under various lighting conditions, improving overall recognition performance.

7.2.2 Exploration of Advanced Models and Loss functions

Advanced and robust models such as DenseNet, ResNeXt, Inception could be used along with more number of datasets. Similarly, model architecture could be revised to utilize triplet loss function and other optimizers like AdaDelta, NADAM could be explored.

7.2.3 Enhanced Security Measures

Future enhancements should prioritize the implementation of robust security measures to mitigate potential threats such as spoofing or presentation attacks. Advanced authentication techniques, including multi-factor authentication and liveness detection, can bolster the system's resilience against adversarial attacks and ensure the integrity of biometric data.

7.2.4 Scalability and Performance Optimization

Efforts to enhance the scalability and performance of the system should focus on optimizing computational efficiency and database management. Strategies such as parallel processing, distributed computing, and database indexing can streamline matching and verification processes, enabling the system to handle larger datasets and accommodate a growing user base without sacrificing performance.

7.2.5 User Experience Enhancements

Improvements to the user experience, including intuitive interfaces, interactive feedback mechanisms, and seamless integration with existing applications, can enhance user adoption and satisfaction. User-centric design principles should be applied to streamline enrollment and authentication processes, making them more user-friendly and accessible to individuals with varying levels of technical proficiency.

8. Conclusion

In conclusion, the development and implementation of the iris recognition system presented in this study marked significant progress in understanding towards biometric security and authentication mechanisms. Through the integration of computer vision techniques, machine learning models, and validation processes, the system has demonstrated promising capabilities in accurately identifying individuals based on their unique iris patterns.

The iris recognition system offers several advantages over traditional authentication methods, including enhanced security, reliability, and user convenience. By leveraging the distinctive features of the iris, such as its stability and uniqueness, the system provides robust protection against identity theft, unauthorized access, and fraudulent activities.

However, despite the system's achievements, several challenges and limitations were encountered during the development and deployment phases. One notable limitation was the inability to achieve real-time iris image capture due to the low resolution of laptop webcams. This constraint hindered the system's performance and scalability, highlighting the importance of investing in specialized hardware components with higher image quality and near-infrared capabilities for future iterations.

Despite these challenges, the iris recognition system holds immense potential for various applications across diverse domains, including access control, border security, healthcare, and financial services. With continued research and development efforts, along with advancements in hardware technology and algorithmic innovations, the system can evolve into a versatile and reliable solution for addressing the evolving security needs of modern society.

Looking ahead, it is important to prioritize ongoing refinement and optimization of the system, including improvements in hardware integration, algorithmic enhancements, security measures, scalability, and user experience. By addressing these areas for improvement, the iris recognition system can realize its full potential as a trusted and effective biometric authentication solution, contributing to a safer and more secure digital ecosystem for individuals and organizations alike.

References

[1] Santosh Giri and Basanta Joshi. Transfer learning based image visualization using cnn. *International Journal of Artificial Intelligence & Applications*, 10(4):47–55, 2019.

[2] Peihua Li, Xiaomin Liu, Lijuan Xiao, and Qi Song. Robust and accurate iris segmentation in very noisy iris images. *Image and vision computing*, 28(2):246–253, 2010.

[3] Zhaofeng He, Tieniu Tan, Zhenan Sun, and Xianchao Qiu. Toward accurate and fast iris segmentation for iris biometrics. *IEEE transactions on pattern analysis and machine intelligence*, 31(9):1670–1684, 2008.

[4] S. Ahmad Radzi, K.-H. Mohamad, S. S. Liew, and R. Bakhteri. Convolutional neural network for face recognition with pose and illumination variation. *International Journal of Engineering and Technology (IJET)*, 6:44–57, 2014.

[5] S. Minaee, A. Abdolrashidiy, and Y. Wang. An experimental study of deep convolutional features for iris recognition. In *Signal Processing in Medicine and Biology Symposium (SPMB), 2016 IEEE*, pages 1–6, 2016.

[6] O. Oyedotun and A. Khashman. Iris nevus diagnosis: Convolutional neural network and deep belief network. *Turkish Journal of Electrical Engineering & Computer Sciences*, 25:1106–1115, 2017.

[7] Sheena S and Sheena Mathew. Cnn based iris recognition system: A novel approach. *Proceedings of the 2nd International Conference on IoT, Social, Mobile, Analytics & Cloud in Computational Vision & Bio-Engineering (ISMAC-CVB 2020)*, 2020. November 23, 2020.

[8] Richard Yew Fatt Ng, Yong Haur Tay, and Kai Ming Mok. An effective segmentation method for iris recognition system. 2008.

[9] Muhammad Arsalan, Hyung Gil Hong, Rizwan Ali Naqvi, Min Beom Lee, Min Cheol Kim, Dong Seop Kim, Chan Sik Kim, and Kang Ryoung Park. Deep learning-based iris segmentation for iris recognition in visible light environment. *Symmetry*, 9(11):263, 2017.

[10] Z Zainal Abidin, M Manaf, AS Shibghatullah, SHA Mohd Yunus, S Anawar, and Z Ayop. Iris segmentation analysis using integro-differential and hough transform in biometric system. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 4(2):41–48, 2012.

[11] Huda Moyasar Therar, Lect Dr Emad Ahmed Mohammed, and Ahmed Jadaan Ali. Multibiometric system for iris recognition based convolutional neural network and transfer learning. In *IOP Conference Series: Materials Science and Engineering*, volume 1105, page 012032. IOP Publishing, 2021.

[12] John Daugman. How iris recognition works. In *The essential guide to image processing*, pages 715–739. Elsevier, 2009.

.1 Appendix A: Code Samples

.1.1 Normalization Code

Preprocessing

```
import cv2 as cv
import os
from matplotlib import pyplot as plt
import numpy as np
import random

main_image = cv.imread(file_path)
gray_image = cv.cvtColor(main_image, cv.COLOR_BGR2GRAY)
median_blurred_image = cv.medianBlur(gray_image, 5)
```

Code Listing 1: Preprocessing Code

Pupil Detection

```
chosen_image=median_blurred_image
circles = cv.HoughCircles(chosen_image, cv.HOUGH_GRADIENT, dp=1, minDist=
   rows/8,
                          param1=100, param2=30,
                          minRadius=20, maxRadius=80)
if(type(circles) == None):
    raise Exception("No circles detected. Error")

if(len(circles)>1):
    raise Exception("Multiple circles detected. Error")

if(len(circles)<1):
    raise Exception("No circles detected. Error")

circles = np.uint16(np.around(circles))

pupil_center = (circles[0,0,0], circles[0,0,1])
pupil_radius = circles[0,0,2]
```

Code Listing 2: Pupil Detection

Iris Localization

```
arr=np.array(chosen_image).reshape(rows,cols).astype(int)
xc,yc=pupil_center[0],pupil_center[1]
radius_range = range(int(pupil_radius*1.4),int(pupil_radius*2.4))
intensity_sum = np.zeros(len(radius_range))
iris_radius = None
max_val = float('-inf')
expected_radius_diff = 40

for i, radius in enumerate(radius_range):
    for theta in range(0, 360):  # Iterate over all angles
        # Convert polar coordinates to Cartesian coordinates
        x = int(xc + radius * np.cos(np.deg2rad(theta)))
        y = int(yc - radius * np.sin(np.deg2rad(theta)))  # Negative sign
   for y-component

        # Ensure the coordinates are within the image bounds
        if 0 <= x < arr.shape[1] and 0 <= y < arr.shape[0]:
            intensity_sum[i] += arr[y, x]

for i in range(2, len(intensity_sum) - 2):
    val = intensity_sum[i + 2] + intensity_sum[i + 1] - intensity_sum[i -
   1] - intensity_sum[i - 2]
    if val > max_val and (radius_range[i] - pupil_radius) >
   expected_radius_diff:
        max_val = val
        iris_radius = radius_range[i]

if (iris_radius-pupil_radius< expected_radius_diff):
    raise ValueError("radius differencd too small")
```

Code Listing 3: Iris Localization

Eyelid Search Region

```
arr=np.array(chosen_image).reshape(rows,cols).astype(int)
length = pupil_radius * 2
height = iris_radius - pupil_radius
start_point_upper = (int(xc - length / 2), yc - iris_radius)
end_point_upper = (int(xc + length / 2), yc - pupil_radius - 5)
start_point_lower = (int(xc - length / 2), yc + iris_radius)
end_point_lower = (int(xc + length / 2), yc + pupil_radius + 5)

# Ensure coordinates are non-negative
```

```
xmin_upper, ymin_upper = start_point_upper
xmax_upper, ymax_upper = end_point_upper
xmin_upper = max(0, xmin_upper)
ymin_upper = max(0, ymin_upper)
xmax_upper = max(0, xmax_upper)
ymax_upper = max(0, ymax_upper)
# For the lower region
xmin_lower, ymax_lower = start_point_lower
xmax_lower, ymin_lower = end_point_lower
```

Code Listing 4: Eyelid Search Region

Custom Sobel Filter

```
def custom_sobel_filter(image, xmin, xmax, ymin, ymax):
    # Initialize Sobel matrix
    sobel_matrix = np.zeros((ymax - ymin, xmax - xmin))

    # Compute Sobel filtering using loop-based implementation
    for i in range(ymin, ymax):
        for j in range(xmin, xmax):
            # Check if indices are within bounds
            if i - 1 >= 0 and j - 1 >= 0 and i + 1 < image.shape[0] and j
   + 1 < image.shape[1]:
                sobel_matrix[i - ymin][j - xmin] = abs(
                    -image[i - 1][j - 1] - 2 * image[i][j - 1] - image[i +
    1][j - 1] +
                    image[i - 1][j + 1] + 2 * image[i][j + 1] + image[i +
   1][j + 1]
                )  # Transposed indexing
            else:
                sobel_matrix[i - ymin][j - xmin] = 0

    return sobel_matrix
```

Code Listing 5: Custom Sobel Filter

Eyelid Segmentation

```
sobel_matrix_upper = custom_sobel_filter(chosen_image, xmin_upper,
   xmax_upper, ymin_upper, ymax_upper)
sobel_matrix_lower = custom_sobel_filter(chosen_image, xmin_lower,
   xmax_lower, ymin_lower, ymax_lower)
```

```
# Compute edge counts
edge_counts_upper = np.sum(sobel_matrix_upper, axis=1)
edge_counts_lower = np.sum(sobel_matrix_lower, axis=1)

# Normalize edge counts
total_edge_count_upper = np.sum(edge_counts_upper)
total_edge_count_lower = np.sum(edge_counts_lower)
edge_counts_normalized_upper = edge_counts_upper / total_edge_count_upper
   if total_edge_count_upper != 0 else np.zeros_like(edge_counts_upper)
edge_counts_normalized_lower = edge_counts_lower / total_edge_count_lower
   if total_edge_count_lower != 0 else np.zeros_like(edge_counts_lower)

# Find eyelid rows
max_edge_upper = np.max(edge_counts_normalized_upper)
max_edge_lower = np.max(edge_counts_normalized_lower)
# print(f"Max Edge Lower Intensity: {max_edge_lower}, Max Edge Upper
   Intensity: {max_edge_upper}")

eyelid_row_upper = ymin_upper + np.argmax(edge_counts_normalized_upper) if
    max_edge_upper >= threshold else yc - iris_radius
eyelid_row_lower = ymin_lower + np.argmax(edge_counts_normalized_lower) if
    max_edge_lower >= threshold else yc + iris_radius

eyelid_row_upper = max(0, eyelid_row_upper)
eyelid_row_lower = min(eyelid_row_lower, image.shape[0] - 1)
```

Code Listing 6: Custom Sobel Filter

Masking

```
mask1 = cv.circle(black_background.copy(), (xc, yc), pupil_radius, 255,
   -1)
mask2 = cv.circle(black_background.copy(), (xc, yc), iris_radius, 255, -1)

# Subtract masks and make into single channel
mask = cv.subtract(mask2, mask1)

# Apply the mask to the original image
result = cv.cvtColor(main_image, cv.COLOR_BGR2GRAY)
result = cv.bitwise_and(result, mask)
```

Code Listing 7: Masking

Custom Rubbersheet Mapping

```
def rubber_sheet_mapping(iris_image, r1, r2,xc,yc, radial_resolution):
    gray = iris_image

    pupil_center = (xc, yc)


    theta_steps = 360  # Number of angular steps
    theta_range = np.linspace(0, 2*np.pi, theta_steps)

    # Initialize normalized iris image
    # normalized_iris = np.zeros((radial_resolution, theta_steps), dtype=
   np.uint8)
    normalized_iris = np.zeros((r2-r1, theta_steps), dtype=np.uint8)
    # Perform rubber sheet mapping
    for r in range(radial_resolution):
        for theta_idx, theta in enumerate(theta_range):
            # Calculate coordinates in polar representation
            x_polar = pupil_center[0] + (r1+r) * np.cos(theta)
            y_polar = pupil_center[1] + (r1+r) * np.sin(theta)
             # Ensure coordinates are within image bounds
            x_polar = max(0, min(gray.shape[1] - 1, int(x_polar)))
            y_polar = max(0, min(gray.shape[0] - 1, int(y_polar)))
            # Interpolate pixel value from original iris image
            normalized_iris[r, theta_idx] = gray[int(y_polar), int(x_polar
   )]

    return normalized_iris
```

Code Listing 8: Custom Rubbersheet Mapping

Normalization

```
iris_image = result

r1 = pupil_radius
r2 = iris_radius

radial_resolution = r2-r1

normalized_iris = rubber_sheet_mapping(iris_image, r1, r2,xc,yc,
```

```
   radial_resolution)
```

Code Listing 9: Normalization

Enhancement

```
clahe = cv.createCLAHE(clipLimit=10.0, tileGridSize=(8, 8))
enhanced_iris = clahe.apply(normalized_iris)
```

Code Listing 10: Enhancement

.1.2 Data Generation Code

Generating a unique id dictionary

```
images = os.listdir('image_folder')
image_groups = {}

for image in images:
    person_id = image.split("_")[0]
    orientation = (image.split("_")[3]).split('.')[0]

    if person_id not in image_groups:
        image_groups[person_id] = {'R':[], 'L':[]}

    image_groups[person_id][orientation].append(image)
```

Code Listing 11: Generating unique ids

Positive pair creation

```
positive_pairs = []
for person_id, image_dict in image_groups.items():
    for orientation, images in image_dict.items():
        pairs = list(itertools.combinations(images, 2))
        positive_pairs.extend(pairs)
```

Code Listing 12: Positive pair creation

Negative pair creation

```
negative_pairs = []
num_positive_pairs = len(positive_pairs)
```

```
positive_pair = random.choice(positive_pairs)

# Get the person IDs in the positive pair
person_id_1 = positive_pair[0].split('_')[1]
person_id_2 = positive_pair[1].split('_')[1]

# # Loop through positive pairs
for i in range(num_positive_pairs):
    # Randomly select a positive pair
    positive_pair = random.choice(positive_pairs)

    # Get the person IDs in the positive pair
    person_id_1 = positive_pair[0].split('_')[0]

    orientation_extract = (positive_pair[0].split('_'))[2]
    orientation_1 = (orientation_extract.split('.'))[0]
    random_choice = random.choice([0,1])
    '''
    randomly select 0 or 1
    0 -> candidate person ID different from person IDs in the positive
   pair
    1 -> candidate person ID is same as person ids in the positive pair
   but has different orientation (left/right)
    '''
    if random_choice==0:
        candidate_person_ids = [person_id for person_id in image_groups.
   keys() if person_id != person_id_1]
        negative_person_id = random.choice(candidate_person_ids)
        random_choice_ag = random.choice([0,1])
        '''
        0 -> choose one image from left
        1 -> choose one image from right
        '''
        if random_choice_ag==0:
            negative_image = random.choice(image_groups[negative_person_id
   ]["L"])
        else:
            negative_image = random.choice(image_groups[negative_person_id
   ]["R"])

    else:
        negative_person_id = person_id_1
```

```
        # If orientation of the positive pair is L then choose R and vice
    versa
        if orientation_1=='L':
            negative_image = random.choice(image_groups[negative_person_id
    ]["R"])
        else:
            negative_image = random.choice(image_groups[negative_person_id
    ]["L"])

     # Add the negative pair to the list
     negative_pairs.append((positive_pair[0], negative_image))
```

Code Listing 13: Negative pair creation