

ABSTRACT

Iris recognition biometric systems have gained significant attention in recent years due to their unparalleled accuracy and reliability in personal identification. This report presents an analysis of iris recognition technology, its underlying principles, and the process followed from start to end. Beginning with the reasonings for why the iris is a preferred biometric in many systems, the report delves into the intricate process of iris image preprocessing, feature extraction, and matching algorithms employed in iris recognition systems. It includes a comparison between the algorithms (Deep Learning, KNN), showing curves and metrics to help decide which blueprint might be considered better. The results show that the Deep Learning Model performs better than KNN.

INTRODUCTION

Iris recognition is the process of recognizing a person by analyzing the random pattern of the iris. It is the colored portion of the eye with coloring based on the amount of melanin pigment within the muscle.

Why Iris?

- 1- **Unique Patterns:** The iris has a highly unique pattern, even more so than fingerprints, making it an excellent candidate for biometric identification. This allows you to differentiate between identical twins.
- 2- **Stability:** Iris patterns remain stable throughout a person's life, unlike some other biometric identifiers which can change over time (like facial features).
- 3- **Low False Acceptance/Rejection Rates:** Iris recognition systems typically have low false acceptance rates (mistakenly accepting an unauthorized person) and low false rejection rates (incorrectly rejecting an authorized person), which are crucial for security applications.
- 4- **Non-invasive:** Iris recognition is non-invasive and does not require physical contact, which enhances user acceptance and hygiene.
- 5- **Fast and Accurate:** Iris recognition algorithms can quickly and accurately match iris patterns, making it suitable for various applications such as access control, border security, and identity verification.
- 6- **Liveness Detection:** Advanced iris recognition systems can incorporate liveness detection to ensure that the presented iris is from a live person, mitigating spoofing attempts using fake irises or images.
- 7- **Unaffected by External Factors:** Iris patterns are generally unaffected by external factors like aging, injuries, or environmental changes, ensuring reliability over time.

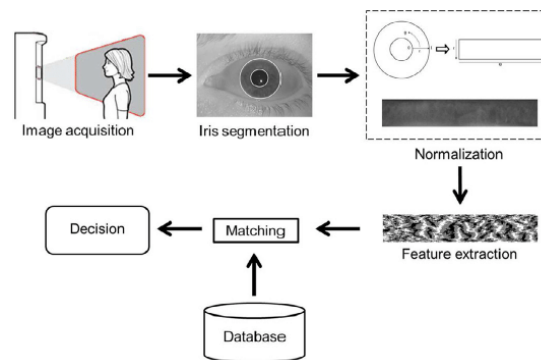
Iris Recognition Block Diagram:

Figure 1: Iris Recognition Block Diagram [2]

The input to the system is an eye image, and the output is an iris template which will provide a mathematical representation of the iris region.

The steps, according to the diagram, are:

- Image Acquisition
- Segmentation
- Normalization
- Feature Extraction / Encoding
- Matching

EQUATIONS:

Polar coordinates:

$$\begin{aligned} x &= \text{center_x} + r \cdot \cos(\theta(i)); \\ y &= \text{center_y} - r \cdot \sin(\theta(i)); \end{aligned} \quad (1)$$

Gabor kernel creation:

```

% Create a Gabor filter kernel
% Adjust the kernel size as needed
kernel_size = 5; % Example: 5x5 kernel
[X, Y] = meshgrid(-floor(kernel_size/2):floor(kernel_size/2));
gabor_kernel = exp(-(X.^2 + Y.^2) / (2 * sigma^2)) .* cos(2 * pi * lambda * X * cos(theta));
  
```

(2)

Gabor bank creation:

```

gabor_bank = cell(num_orientations, num_scales);
for o = 1:num_orientations
    for s = 1:num_scales
        theta = (o - 1) * pi / num_orientations;
        % Create a Gabor filter kernel
        gabor_kernel = create_gabor_kernel(lambda, theta, sigma);
        gabor_bank{o, s} = gabor_kernel;
    end
end
  
```

(3)

Statistical Measures calculation:

```
mean_val = mean(filtered_responses(:));
variance_val = var(filtered_responses(:));
energy_val = sum(filtered_responses(:).^2);
entropy_val = -sum(filtered_responses(:) .* log2(filtered_responses(:) + eps));
```

(4)

EER:

```
far = fpr;
frr = (1-tpr);
[mindistance, indexofmin] = min(abs(far-frr));
eer_far = far(indexofmin);
eer_frr = frr(indexofmin);
eer = ((far(indexofmin) + frr(indexofmin)) / 2);
```

(5)

:

D Prime:

```
meangen = mean(genuine_scores);
meanimp = mean(imposter_scores);
sigmagen = std(genuine_scores);
sigmaimp = std(imposter_scores);
d_prime = (sqrt(2) * abs(meangen - meanimp)) / (sqrt((sigmagen^2 + sigmaimp^2)));
```

(6)

Rank 1 Identification Rate:

```
genuine_scores = genuine_scores(:);
flooredgen = floor(genuine_scores);
threshold = mode(flooredgen);
highscores = sum(flooredgen >= 70);
GenLength = length(genuine_scores);
rank1rate = ( highscores / GenLength );
```

(7)

TMR@FMR:

```
[~, idx_1] = min(abs(fpr - 0.01));
[~, idx_001] = min(abs(fpr - 0.0001));
tmr_1 = tpr(idx_1);
tmr_001 = tpr(idx_001);
```

(8)

DATABASE DESCRIPTION

Consists of 35 subjects. Each subject has 15 training images and 4 testing images.
Images are grey scaled and segmented.

METHOD DESCRIPTION

- **Preprocessing:** [\[4\]](#)
 - **Normalization using Daugman.**

Normalization using Daugman's method in an iris recognition system involves transforming the iris image into a standardized form, ensuring consistency and comparability across different iris samples. This normalization process helps mitigate variations in iris appearance due to factors such as pupil dilation, occlusion, and imaging conditions. Here is a description of the normalization process using Daugman's method:

1. **Iris Localization:** Done using Hough Transform.
2. **Polar Coordinate Transformation:** Once the iris region is localized, Daugman's normalization method transforms the Cartesian coordinates of the pixels within the iris region into polar coordinates. This transformation maps each pixel's position in the iris to its corresponding angular position and radial distance from the iris center.

```
% Generate polar coordinates grid
[X, Y] = meshgrid(1:cols, 1:rows);
X = X - center_x;
Y = center_y - Y;

polar_iris = zeros(rows, cols, 'double');
for i = 1:cols
    % Compute polar coordinates for this angle
    x = center_x + r .* cos(theta(i));
    y = center_y - r .* sin(theta(i));
```

Figure 2.1: Computation of polar coordinates in MATLAB using equation (1)

3. **Interpolation:**

```
for i = 1:cols
    % Compute polar coordinates for this angle
    x = center_x + r .* cos(theta(i));
    y = center_y - r .* sin(theta(i));

    % Interpolate values from Cartesian to polar coordinates
    polar_values = interp2(double(iris_img), x, y, 'linear', 0);

    % Assign interpolated values to polar image
    polar_iris(:, i) = polar_values;
end
```

Figure 2.2: Interpolation process in MATLAB

4. **Unwrapping:**

The rubber sheet model assigns to each point in the iris, a pair of dimensionless real coordinates r , θ where r lies in the unit interval $[0, 1]$ and θ is the angular variable, cyclic over $[0, 2\pi]$.

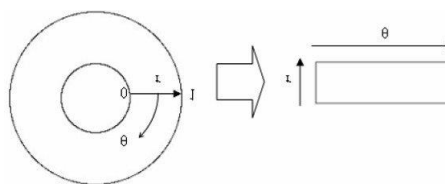


Figure 2.3: Unwrapping of iris and normalization

5. Normalization:

Normalization is then applied to compensate for variations in iris size, shape, and illumination.

```
% Normalize polar image
polar_iris = mat2gray(polar_iris);

% Convert back to uint8
out_img = uint8(255 * polar_iris);
```

Figure 2.4: Normalization in MATLAB

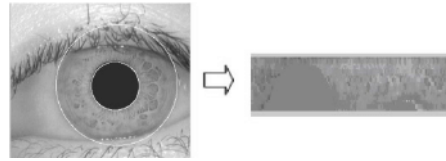


Figure 2.5: Normalized iris

- **Feature Extraction:**

- **2D Gabor Filter**

The 2D Gabor filter is widely used in image processing and computer vision for feature extraction because of its ability to capture localized spatial frequency information in different orientations and scales. The normalized image from Daugman is used to get the output.

- **Filtering:**

Multiple Gabor filters with different orientations, scales, and aspect ratios are typically used to capture various texture patterns present in the iris. Each Gabor filter is convolved with the normalized iris image (See figures 3.1 – 3.3). The convolution operation produces response maps highlighting regions of the iris that are similar to the filter's orientation and frequency characteristics.

```
function gabor_kernel = create_gabor_kernel(lambda, theta, sigma)
% Create a Gabor filter kernel
% Adjust the kernel size as needed
kernel_size = 5; % Example: 5x5 kernel
[X, Y] = meshgrid(-floor(kernel_size/2):floor(kernel_size/2));
gabor_kernel = exp(-(X.^2 + Y.^2) / (2 * sigma^2)) .* cos(2 * pi * lambda * X * cos(theta));
end
```

Figure 3.1: Creation of Gabor Kernel using equation (2)

```
% Create Gabor filter bank (same as before)
% ...
gabor_bank = cell(num_orientations, num_scales);
for o = 1:num_orientations
    for s = 1:num_scales
        theta = (o - 1) * pi / num_orientations;
        % Create a Gabor filter kernel
        gabor_kernel = create_gabor_kernel(lambda, theta, sigma);
        gabor_bank{o, s} = gabor_kernel;
    end
end
```

Figure 3.2: Creation of Gabor Filter Bank (many filters with different orientations) using equation (3)

```
% Apply Gabor filters to the normalized iris image (same as before)
% ...
filtered_responses = zeros(size(normalized_iris, 1), size(normalized_iris, 2), num_orientations * num_scales);
for o = 1:num_orientations
    for s = 1:num_scales
        filtered_responses(:, :, (o - 1) * num_scales + s) = ...
            imfilter(normalized_iris, gabor_bank{o, s}, 'symmetric');
    end
end
```

Figure 3.3: Application of Gabor filters

Compute statistical measures for the filtered images.

```
% Compute statistical measures for the entire filtered response
mean_val = mean(filtered_responses(:));
variance_val = var(filtered_responses(:));
energy_val = sum(filtered_responses(:).^2);
entropy_val = -sum(filtered_responses(:) .* log2(filtered_responses(:) + eps));
```

Figure 3.4: Calculation of mean, variance, energy, entropy using equation (4)

- **Matching**

The matching process includes loading the pre-trained models, performing normalization and using the statistical measures calculated to predict the class.

```
load('KNN_model.mat', 'knnModelBest');
csvFilePath = "D:\year3\biometrics\Project_final\feature_extraction.csv";
data = xlsread(csvFilePath);
```

Figure 3.5: Loading KNN model and features.

```
load('tree_model.mat', 'mdl');
csvFilePath = "D:\year3\biometrics\Final_project\feature_extraction.csv";
data = xlsread(csvFilePath);
```

Figure 3.6: Loading Trees model and features.

```
normalized_image=Normalization_Daugman(img);
[mean_val, variance_val, energy_val, entropy_val] = Gabor_features_extractor1(normalized_image);
[knnPredictedLabels, knnScores]= predict(knnModelBest, [mean_val, variance_val, energy_val, entropy_val]);
class = knnPredictedLabels;
```

Figure 3.7: Matching Process KNN

```
normalized_image=Normalization_Daugman(img);
[mean_val, variance_val, energy_val, entropy_val] = Gabor_features_extractor1(normalized_image)
[treeLabels, treeScores]= predict(mdl, [mean_val, variance_val, energy_val, entropy_val]);
class = treeLabels;
```

Figure 3.8: Matching Process Decision Trees

• Results and Discussion

Accuracy		
KNN	DEEP LEARNING	Decision Trees
62.8%	96.88%	98.36%

Table 1: Accuracy comparison between KNN, Decision Trees and Deep Learning Models

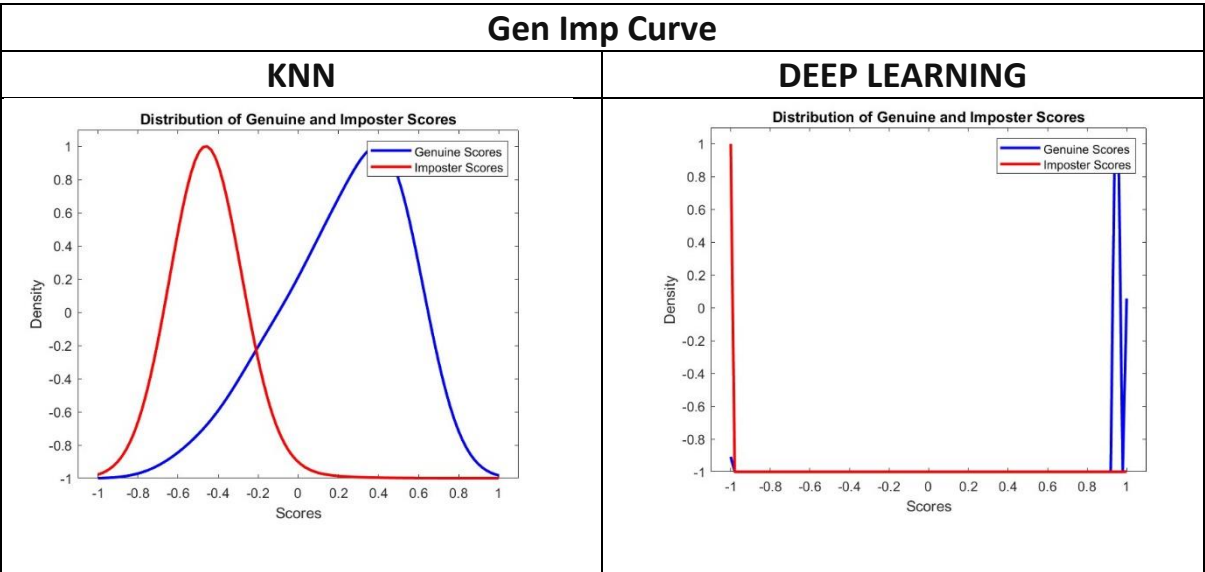


Table 2: Gen Imp Curve comparison between KNN and Deep Learning Models

Using KNN, the genuine and imposter curves intersect showing a higher percentage of False Matches (imposters seen as genuine) and False Non-Matches (genuines shown as imposters). However, they do not intersect when using Deep Learning, showing a lower percentage of False Matches and False Non-Matches.

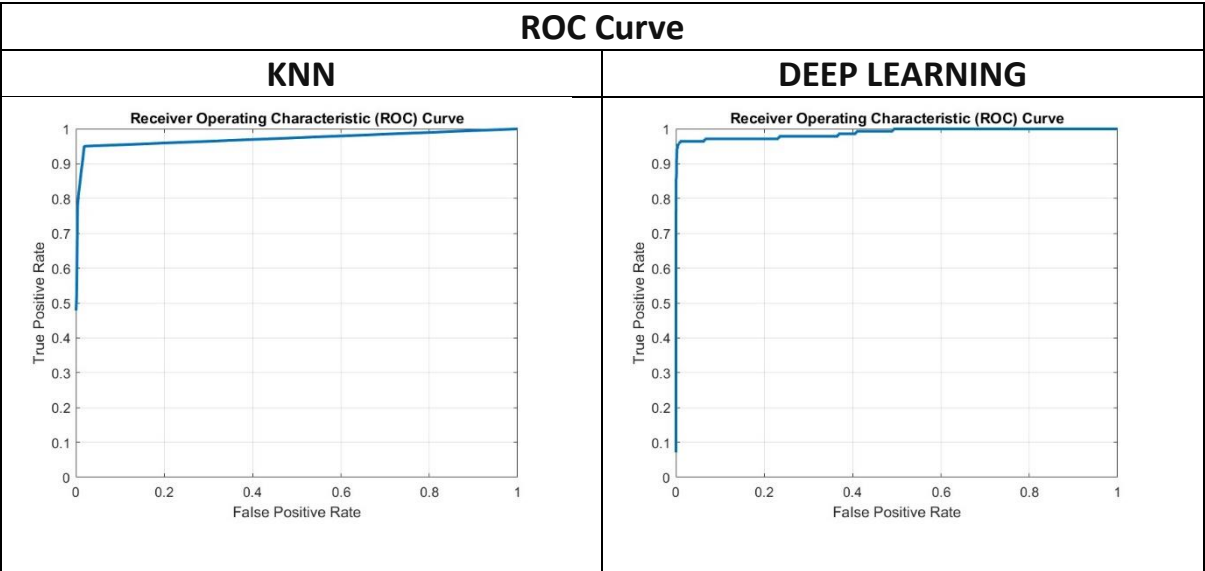


Table 3: ROC comparison between KNN and Deep Learning Models

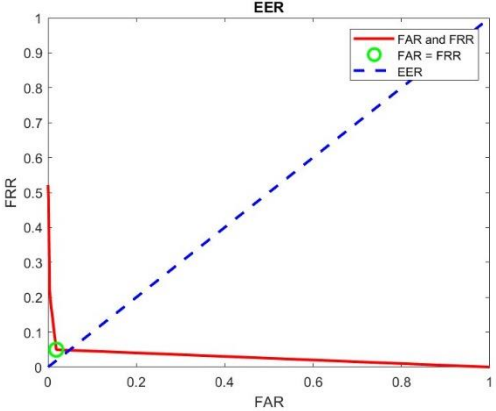
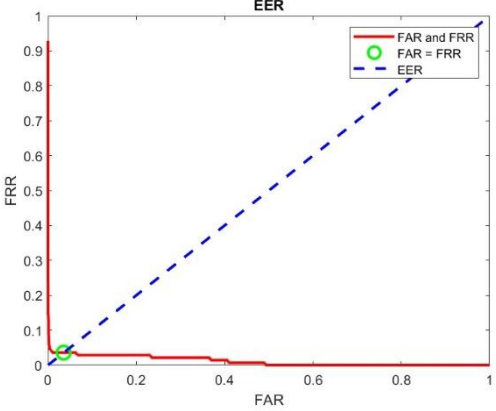
EER	
KNN	DEEP LEARNING
	
EER_FAR: 0.018787012456606 EER_FRR: 0.050000000000000 EER: 0.034393506228303	EER_FAR: 0.035531958341842 EER_FRR: 0.035714285714286 EER: 0.035623122028064

Table 4: EER values comparison between KNN and Deep Learning Models. Uses equation [\(5\)](#)

D-Prime	
KNN	DEEP LEARNING
3.364265797976948	5.375694840954670

Table 5: D Prime comparison between KNN and Deep Learning Models. Uses equation [\(6\)](#)

A higher D prime value implies a higher performance. D-Prime provides a quantitative measure of the effectiveness of a biometric system in correctly distinguishing between genuine and impostor matches.

RANK 1 IDENTIFICATION RATE	
KNN	DEEP LEARNING
0.478571428571429	0.728571428571429

Table 6: Rank 1 Identification Rate comparison between KNN and Deep Learning Models. Uses equation [\(7\)](#)

Rank 1 identification rate refers to the accuracy of identifying an individual when only considering the top-ranked matches provided by the biometric system. A higher Rank 1 Rate shows a higher performance.

TMR@FMR (1% , 0.01%)			
KNN		DEEP LEARNING	
1%	0.01%	1%	0.01%
0.814285714285	0.478571428571	0.964285714285	0.764285714285

Table 7: TMR comparison between KNN and Deep Learning Models. Uses equation [\(8\)](#)

A TMR of 0.xx at FMR of 1% means that when the system operates with a False Match Rate (FMR) of 1%, it correctly identifies genuine matches xx% of the time. Using the previous relation, Deep Learning correctly identifies genuine matches 96% of the time at FMR of 1% and 76% of time at FMR of 0.01%, while KNN correctly identifies genuine matches 81% of the time at FMR of 1% and 47% of time at FMR of 0.01%.

CONCLUSION:

It can be concluded that Deep Learning Model performs better than KNN not only from the accuracy achieved by Deep Learning, but also from the performance curves and metrics measured and shown above.

REFERENCES

- [\[1\]](#) Jasem Rahman Malgheet, Noridayu Bt Manshor, Lilly Suriani Affendey, "Iris Recognition Development Techniques: A Comprehensive Review", Complexity, vol. 2021, Article ID 6641247, 32 pages, 2021.
- [\[2\]](#) A Brief Survey on Modern Iris Feature Extraction Methods , International Journal of Engineering and Technology 39(1A):123-129, January 2021
- [\[3\]](#) Iris Recognition by Prof. John Daugman
- [\[4\]](#) IRIS RECOGNITION
- [\[5\]](#) EEL6825 Iris Recognition