Iris Segmentation and Recognition
Final Report

June 28, 2021

Contents

Abstract

Iris recognition is an important biometric identifier. This report offers multiple techniques to handle the problem of both iris segmentation and recognition. The report can be divided into two parts, the first is applying classic computer vision techniques for both segmentation and recognition and the second is achieving the same goal using deep learning. For developing and testing our systems, we used the dataset *IITD* [KP10] which has 224 subjects and 10 images per subject. A comparison then of the techniques along with their score ratings is available in the report. It is worth noting that there was a challenge with the dataset that the samples per subject are very similar to each other and the normal splitting procedures won't be a real test of generality, so we tried to use different data splitting techniques whenever possible and we report the difference.

1 Dataset Exploration

The training and testing data comes from the IITD dataset consisting of 2240 images of the iris and their corresponding masks captured using JIRIS, JPC1000, digital CMOS camera in the same settings. All the images are in bitmap ($*$ .bmp) format. All the subjects in the database are in the age group 14-55 years comprising 176 males and 48 females. Each person is also identified as separate as there is a naming scheme being followed. For images, they are in a separate folder with a name as an id for that individual and 'L' or 'R' representing the left or right eye. For masks, they named it 'OperatorA_001-A_01', where A represents the right eye of 001 individual identities and is the first image. Following are some of the randomly chosen sample images from the dataset in figure 1.

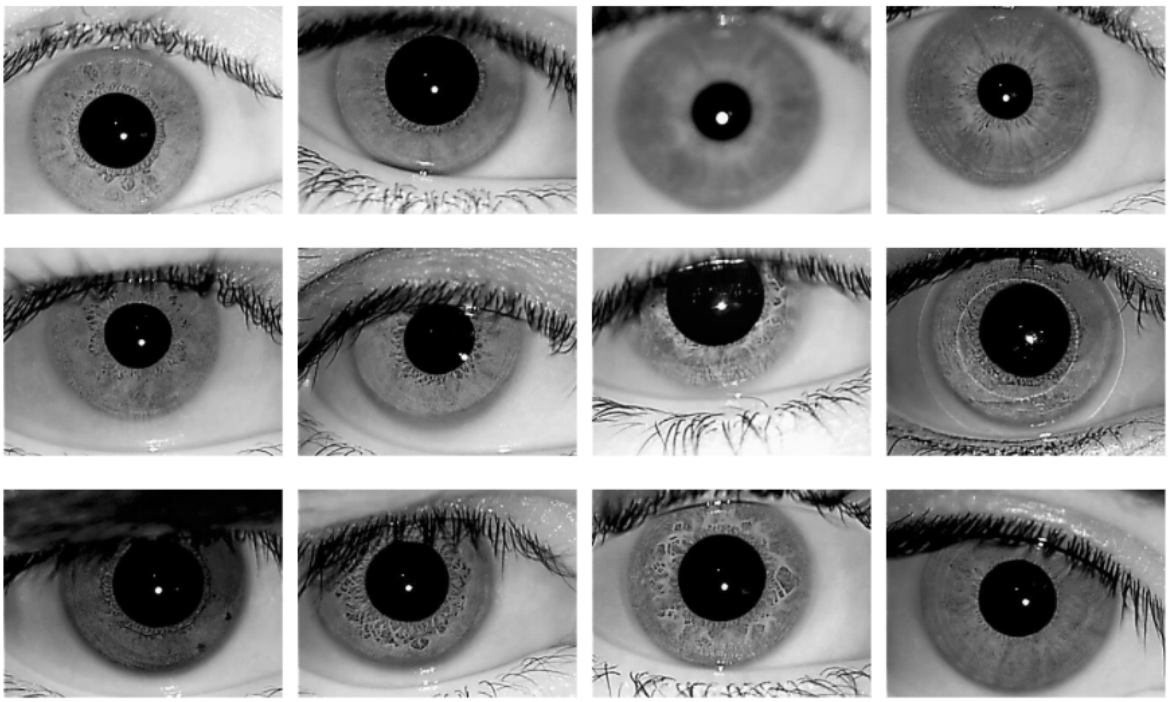

Figure 1: Random Samples from the Dataset

1.1 Challenging Samples

There were several challenges to this dataset. For recognition, the deep learning-based approaches suffered because there was very few samples per class and a large number of classes, 224. Also the distributions of the user's samples are very similar which makes generalization testing very difficult. We handled this, when possible by splitting the samples into train and test by class, meaning that for each of the 224 classes we randomly select 122 classes as train and the same for testing for deep learning method, and for classical approach we took 5 images of each class for training and other 5

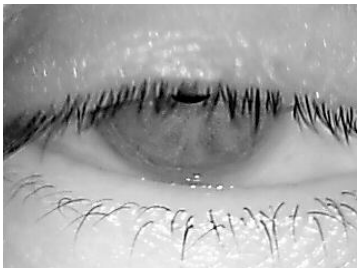

Figure 2: Example of Hard Sample, Image 02_L of 87^{th} Subject

Figure 3: Easy vs Relatively Harder Example

for testing. Using this split the model will encounter relatively new data different from the dataset and we can test if the model overfit or it generalized well. Also, in some cases, the iris was partially occluded and some viewpoints were different as shown in figure 1, others were relatively difficult and were compared in figure 3, while others are extreme cases as shown in figure 2. We addressed and overcame to achieve good results.

2 Image processing

This part concerning applying image processing techniques for image segmentation and pre-processing for the following steps in the recognition algorithm.

It is worth noting the the dataset is considered very clean so no image enhancement techniques were needed, also the images were centered on the iris and have the same shape so no pre-processing was needed there.

2.1 Iris segmentation

In our search for an appropriate algorithm for iris segmentation many methods have been tested. we explored clustering algorithms, geometric algorithms such as *Hough transform* due to the fact that the iris is almost circular

The main problem for the clustering methods is the fact that in many cases there are no clear borders between iris and sclera, as you can see on the 4. Left part of the iris blends into sclera making these four algorithms fail to differentiate areas of iris.

On the contrary, the main disadvantage of geometric approach of Hough transform is that eyelid falls into segmented iris circle. In 5 you can notice how top eyelid enters the iris Hough circle.

In our approach, we have decided to combine geometric and clustering methods to improve the segmentation results. For identifying iris and pupil borders *Daugman's integrodifferential operator* (IDDO) for Hough transform [Dau09] was used. For the clustering *K-means* and *Watershed* were implemented in-between borders of pupil and iris.

For K-means, we used $k = 2$, to segment iris vs everything else; however for watershed we had to come up with the marker strategy. After inspecting the data set, we realized that top and bottom parts of irises usually could be covered up, but not the left and right parts. Thus, we set each line markers of 5 pixels from the left and right sides of pupil contour, found by Hough algorithm. The length of lines are varied to the radius of iris and pupil as shown in figure 6 .

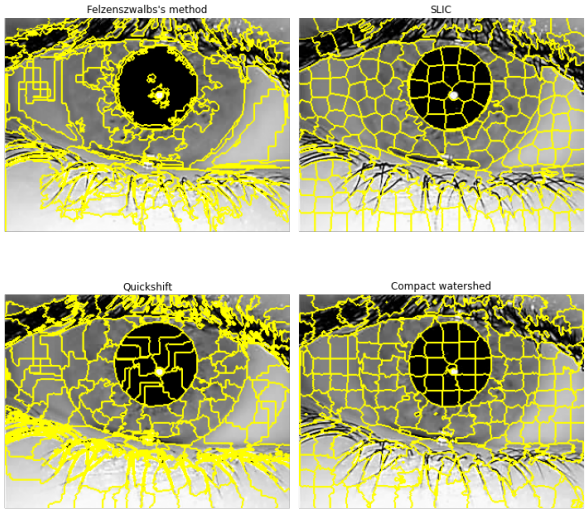

Figure 4: Clustering Methods for Segmentation

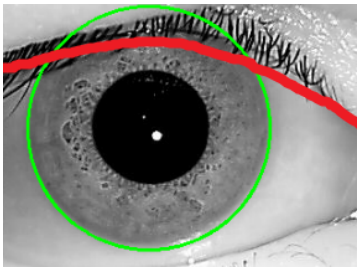

Figure 5: Iris Hough Circle and Top Eyelid Intersection

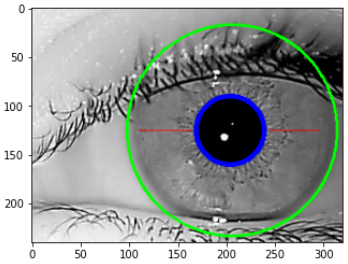

Figure 6: Segmentation strategy. Green line represents iris contour, blue - pupil, red line shows the marker for watershed.

Equation 1 shows the heuristics used for computing the marker's length

$$L_{marker} = 4/5 * (R_{iris} - R_{pupil}) \tag{1}$$

The segmentation f1 score for simple circle Hough transform using *IDDO*, Hough transform in addition to K-means (referred as just K-means) and Hough transform in addition to watershed segmentation (referred as just watershed) are shown in figure 1 and examples could be seen in figure 7

2.2 Iris Normalization

For more effective iris recognition, technique called 'normalization' is used. Using parameters of circles, such as radii and center positions of pupil and iris, we can warp *donut-like* shape of the iris into rectangle as you can see in figure 8. The final shape of the normalized image was set to be 20×240.

Segmentation method	F1 score
Hough	0.815
Watershed	0.848
K-means	0.738

Table 1: F1 Scores for Segmentation.

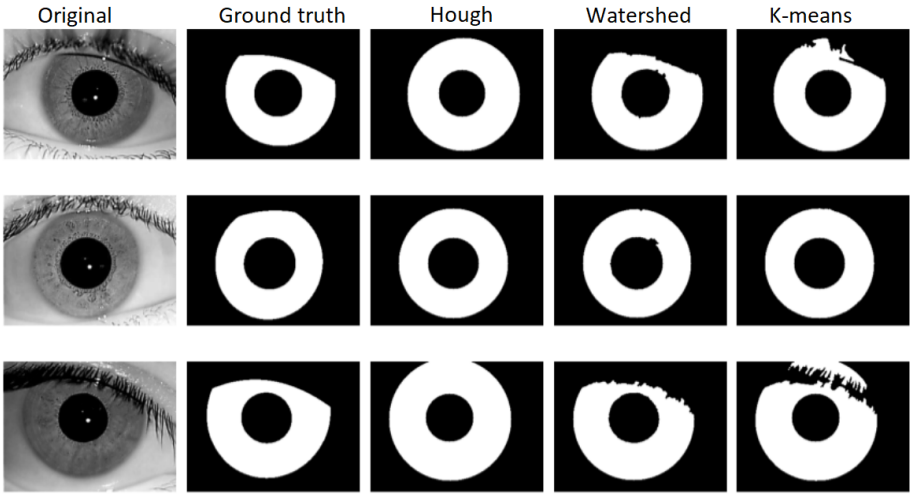

Figure 7: Comparisons of Segmentation Methods on three Image Samples

3 Feature Extraction and Machine Learning Iris Recognition

This section is concerned with *encoding* the normalized image into a feature-like vector which is considered as an input for the machine learning module, each encoding technique is followed by the machine learning techniques used for training on the encodings.

All the hyper-parameters used by the feature extraction algorithms are chosen with the machine learning algorithms' hyper-parameters using $5 - Fold\ Cross\ Validation$, the average of the cross validation results will be pointed out in tables as CV. The cross validation made the tuning a harder process, in terms of computational resources and execution time but the resulting scores are considerably reliable.

It is worth noting that due to the nature of the dataset and the data splitting we need for training the recognition models, overfitting both the train and test dataset is an inescapable fact that we need to deal with as much as possible while interpreting the results of the models following.

3.1 Gabor Filters

Gabor filters are used because they are the mostly used feature extraction in the literature for iris recognition. Part of its popularity is from the fact that the output is binary and the following recognition step is KNN with $k = 1$ and the distance function applied is hamming distance (in our case decidability index of 0.66, test ROC = 0.94 and accuracy = 0.88), so all the distance computations can be modeled as binary operations which are lighting fast which is a feature used if the system is comparing the distance with the population of a country such as the recognition system applied in India right now. The output of the gabor encoding is binary matrix, figure 10.

In table 2, the results of different machine learning algorithms are shown, where CV is the average of the 5 folds used for training.

From table 2, we see an overall very high AUC score for both testing and training. which we interpret as an overfitting of the features but we have no means to fix or quantify this problem.

Since gabor output is just binary matrix, we decimilized the gabor outputs and used same model to compare results (Table 3).

Even though training for binary data was significantly longer, it showed better results.

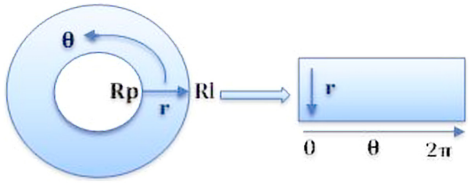

Figure 8: Normalization schema. Source: [RAA+19]

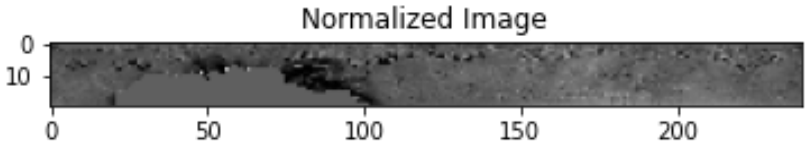

Figure 9: Example of normalized iris

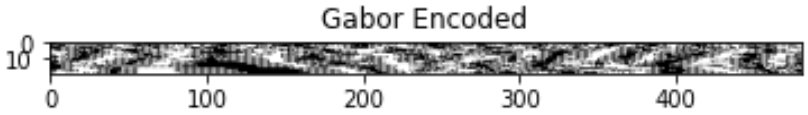

Figure 10: Example of gabor encoded iris

Model Name	Train AUC	Train EER	Test AUC	Test EER	Best parameters
Logistic Regression	1	0.484	0.994	0.485	{'C': 50, 'penalty': 'l2'}
Desicion Tree	1	0.004	0.563	0.004	{'max_depth': 21}
Random Forest	1	0.508	0.977	0.499	{'max_depth': 11, 'n_estimators': 1800}
Adaboost	1	0.489	0.964	0.485	{'learning_rate': 1.4, 'n_estimators': 2000}
SVC	0.995	0.488	0.979	0.492	{'C': 20, 'kernel': 'sigmoid'}
GaussianNB	1	0.011	0.982	0.405	{'var_smoothing': 2}
KNN	1	0.004	0.96	0.004	{'n_neighbors': 1}

Table 2: Results of classical machine learning approach on binary gabor data.

Model Name	Train AUC	Train EER	Test AUC	Test EER	Best parameters
Logistic Regression	1	0.498	0.982	0.505	{'C': 0.1, 'penalty': 'l2'}
Desicion Tree	1	0.004	0.548	0.004	{'max_depth': 181}
Random Forest	1	0.493	0.898	0.492	{'max_depth': 11, 'n_estimators': 1800}
Adaboost	0.692	0.673	0.611	0.644	{'learning_rate': 1.3, 'n_estimators': 300}
SVC	0.972	0.494	0.962	0.516	{'C': 20, 'kernel': 'linear'}
GaussianNB	1	0.517	0.977	0.497	{'var_smoothing': 0.811}
KNN	0.997	0.03	0.944	0.032	{'n_neighbors': 9}

Table 3: Results of classical machine learning approach on decimalized gabor data.

3.2 Local Binary Patterns (LBP)

LBP is a global image descriptor. And from the nature of the iris, that the patterns are more localized and focused in small neighborhoods, LBP was used to capture these patterns and encode them. The implementation used is from *scikit-image* and the output is a feature image of size 20×240. The feature image histogram with 256 bins is computed and this is considered the feature vector that will be used for the recognition.

The main parameter used for tuning is the *radius* of the LBP kernel and accordingly the number

of points is computed as shown in equation 2. This equation just empathise that as we increase the radius, the number of points in the neighborhood should also increase.

$$points = 8 * radius \tag{2}$$

Three radius values are used to cover different scales then we concatenate the different feature vectors from different radius values in order to capture multi-scale information in one feature vector. The different methods are shown with their code name and the decidablity index in table 4. The code name will be referred to in the machine learning part.

Code Name	Radius	Number of Points	feature vector size	DI
LBP_{r1}	1	8	256	0.775
LBP_{r3}	3	24	256	0.668
LBP_{r5}	5	40	256	0.644
LBP_{concat}	$[1, 3, 5]$	$8 + 24 + 40$	$256 * 3 = 768$	0.696

Table 4: Feature Extraction with LBP

The comparative summary of the machine learning methods used is shown in table 5. Some notes on the training are the following:

- Between the three values for the radius used, LBP_{r3} is the best overall.
- The concatenation of all the feature vectors is the best overall, which makes sense because we are using multi-scale data but the trade off is between the bigger feature vector and the increased computation and inference time.
- PCA achieved good results meaning that the LBP features can be reduced to simpler representation and yet get high results.
- Filter feature selection methods failed to get better results unlike PCA. That points out that the features can be represented in more compact form but need more complex engineering to get the compact form.
- Wrapper feature selection methods such as *recursive feature selection* worked nicely with random forest to be good results but just the random forest was able to get higher results empathizing on the previous point.
- The best performing model overall is the Random Forest

3.3 Multi-Block Local Binary Pattern

Although LBP showed good results using different machine algorithms but using another version of LBP we can control the window where LBP is being computed gaining more flexibility in choosing the window size for the LBP. For this end we use *Multi-block LBP (MB-LBP)*.

Table 6 shows the parameters used for this algorithm. These parameters are chosen to explore different neighborhoods in the normalized image. The table 7 shows the result of applying different machine learning algorithms, since they have very low dimension in the first place, feature engineering is not necessary.

Training conclusions:

- Any feature selection method gets worst results with respect to the original data given the usage of the same model. This makes sense since the features themselves have low dimensions.
- The best models are *Gradient Boost* and *Random Forest*
- These features are better and more compact than the usage of normal LBP

Algorithm	Code Name	CV ROC	Test ROC	Train EER	Test EER
Logistic Regression (LG)	LBP_{r1}	0.85	0.86	0.13	0.19
	LBP_{r3}	0.885	0.886	0.11	0.16
	LBP_{r5}	0.795	0.827	0.15	0.22
	LBP_{concat}	0.89	0.91	0.06	0.14
LG with PCA	LBP_{r1}	0.92	0.928	0.05	0.12
	LBP_{r3}	0.89	0.893	0.07	0.17
	LBP_{r5}	0.873	0.874	0.08	0.18
	LBP_{concat}	0.942	0.946	0.04	0.09
LG with Feature Selection	LBP_{r1}	0.845	0.828	0	0.23
	LBP_{r3}	0.76	0.71	$3 * 10^{-3}$	0.3
	LBP_{r5}	0.74	0.59	0.0018	0.4
	LBP_{concat}	0.836	0.861	0	0.2
Random Forest	LBP_{r1}	0.925	0.929	0	0.12
	LBP_{r3}	0.929	0.937	0	0.1
	LBP_{r5}	0.921	0.937	0	0.1
	LBP_{concat}	0.94	0.948	0	0.1
RFE with Random Forest	LBP_{r1}	0.915	0.917	0	0.12
	LBP_{r3}	0.923	0.93	0	0.1
	LBP_{r5}	0.912	0.926	0	0.13
	LBP_{concat}	0.93	0.935	0	0.08
KNN	LBP_{r1}	0.66	0.685	0.01	0.017
	LBP_{r3}	0.6	0.6	0.13	0.01
	LBP_{r5}	0.59	0.59	0.014	0.017
	LBP_{concat}	0.679	0.689	0.012	0.016
Adaboost	LBP_{r1}	0.675	0.639	0.4	0.44
	LBP_{r3}	0.699	0.71	0.33	0.37
	LBP_{r5}	0.63	0.724	0.27	0.33
	LBP_{concat}	0.63	0.66	0.45	0.44
Gradient boost	LBP_{r1}	0.86	0.88	$4 * 10^{-}4$	0.18
	LBP_{r3}	0.85	0.88	0	0.19
	LBP_{r5}	0.858	0.87	0	0.19
	LBP_{concat}	0.89	0.915	0	0.15

Table 5: Iris Recognition on LBP Features

Code Name	Block Width	Block Height	Feature Vector Size	DI
$MB - LBP_1$	10	2	32	1.055
$MB - LBP_2$	5	3	48	1.14

Table 6: Feature Extraction with MB-LBP

Algorithm	Code Name	CV ROC	Test ROC	Train EER	Test EER
Logistic Regression (LG)	$MB-LBP_1$	0.92	0.936	0	0.12
	$MB-LBP_2$	0.9	0.91	2.4	0.16
LG with PCA	$MB-LBP_1$	0.878	0.89	0.08	0.16
	$MB-LBP_2$	0.868	0.878	0.1	0.18
LG with Feature Selection	$MB-LBP_1$	0.819	0.81	0.16	0.24
	$MB-LBP_2$	0.8	0.79	0.14	0.27
Random Forest	$MB-LBP_1$	0.95	0.97	0	0.05
	$MB-LBP_2$	0.956	0.968	0	0.06
RFE with Random Forest	$MB-LBP_1$	0.943	0.954	0	0.05
	$MB-LBP_2$	0.947	0.958	0	0.06
KNN	$MB-LBP_1$	0.81	0.82	0.01	0.01
	$MB-LBP_2$	0.8	0.82	0.11	0.15
Adaboost	$MB-LBP_1$	0.77	0.745	0.19	0.31
	$MB-LBP_2$	0.84	0.898	0.006	0.17
Gradient boost	$MB-LBP_1$	0.94	0.96	0	0.07
	$MB-LBP_2$	0.95	0.956	0	0.08

Table 7: Iris Recognition on MB-LBP Features

3.4 Histogram of Gradients (HoG)

HoG was chosen for the same reason as *LBP* that it is a local feature extractor but with a fine control over the shape of the feature vector for the entire image. The parameters used for the algorithm are shown in table 8

Code Name	Orientation	Pixels per Cell	Feature Vector Size	DI
HOG_4	4	(6, 10)	792	1.064
HOG_8	8	(6, 10)	1584	0.882
HOG_{16}	16	(6, 10)	3168	0.723

Table 8: Feature Extraction with HoG

A theory was tested here that some parts of the image are more informative than the others using different feature selection techniques. Since *HoG* has more features than *LBP* and *MB-LBP* also, each feature is very local so if feature selection methods work then we can say that some parts of the images are better than the others.

The results are shown in table 9. Here we started with different feature vector sizes and we applied both feature selection and *PCA* as dimensionality reduction and the conclusions as following:

- PCA got results comparable to the original data but with very high number of principle components showing that the data is hard to compress any further
- Feature selection got results only for very high number of features showing that we can't remove some of the features.
- The best model is simple linear regression.
- The best feature extraction method so far is *HoG*

3.5 Final Notes

We need to point out that the high performance of the different models is most likely because of the overfitting of the data and we tried to explore different theories and better representations of the output both in classical feature extraction and deep learning.

Also Machine learning algorithms were applied on the output of our representation learning model which gave good results.

The best pipeline is using *HoG* as feature extraction then classifying using logistic regression.

Algorithm	Code Name	CV ROC	Test ROC	Train EER	Test EER
Logistic Regression (LG)	HOG_4	0.976	0.986	0	0.025
	HOG_8	0.979	0.989	0	0.02
	HOG_{16}	0.976	0.986	0	0.025
LG with PCA	HOG_4	0.975	0.979	0	0.024
	HOG_8	0.977	0.983	0	0.03
	HOG_{16}	0.974	0.98	0	0.029
LG with Feature Selection	HOG_4	0.966	0.977	0	0.04
	HOG_8	0.96	0.977	0	0.04
	HOG_{16}	0.93	0.945	0	0.9
Random Forest	HOG_4	0.96	0.96	0	0.06
	HOG_8	0.94	0.947	0	0.08
	HOG_{16}	0.91	0.927	0	0.12
RFE with Random Forest	HOG_4	0.918	0.933	0	0.005
	HOG_8	0.88	0.9	0	0.09
	HOG_{16}	0.847	0.869	0.0001	0.14
KNN	HOG_4	0.91	0.93	0.005	0.008
	HOG_8	0.92	0.94	0.005	0.008
	HOG_{16}	0.92	0.94	0.005	0.008
Adaboost	HOG_4	0.53	0.59	0.63	0.65
	HOG_8	0.56	0.62	0.48	0.5
	HOG_{16}	0.5	0.51	0.88	0.9
Gradient boost	HOG_4	0.96	0.97	0	0.05
	HOG_8	0.94	0.95	0	0.08
	HOG_{16}	0.924	0.924	0	0.13

Table 9: Iris Recognition on HoG Features

4 Semantic Segmentation

For deep learning, we tried several things. The basic pipeline for deep learning has two steps. First, we need to segment the iris images than using the masks and images to perform classification or recognition to identify the individuals.

It should be noted that we use different data splitting than the machine learning part. Here when possible we select at random 50% of the classes to be the training dataset while the other 50% are the test set and we compare the both data splitting techniques to test if our models can actually generalize well over the test set.

Semantic segmentation means giving a class label but at the pixel level. So in our case,we need to segment out the iris region from the background. So we have to perform binary class segmentation to feed our pipeline. For this, we tried different architectures with multiple configurations.

- Baseline convolutional neural network
- U-Net architecture
- U-Net with attention

4.1 Baseline Convolutional Neural Network

This model shown in figure 11 consisted only of convolutional layers and Relu activations after each layer. This was made for proof of concept and to check if simple segmentation works. Surprisingly this gave quite good results. It was mainly because our training and test data had many similarities even though the data was less. As in this setting, we split it with a fifty-fifty ratio for each individual.

In this model, we only changed the kernel sizes and paddings accordingly to keep the input and output dimensions of height and width the same but kept on changing the number of kernels to vary the number of channels.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 128, 212, 212]	46,336
ReLU-2	[-1, 128, 212, 212]	0
Conv2d-3	[-1, 320, 212, 212]	41,280
ReLU-4	[-1, 320, 212, 212]	0
Conv2d-5	[-1, 320, 212, 212]	102,720
ReLU-6	[-1, 320, 212, 212]	0
Conv2d-7	[-1, 320, 212, 212]	102,720
ReLU-8	[-1, 320, 212, 212]	0
Conv2d-9	[-1, 128, 212, 212]	41,088
ReLU-10	[-1, 128, 212, 212]	0
Conv2d-11	[-1, 128, 212, 212]	147,584
ReLU-12	[-1, 128, 212, 212]	0
Conv2d-13	[-1, 512, 212, 212]	66,048
ReLU-14	[-1, 512, 212, 212]	0
Conv2d-15	[-1, 128, 212, 212]	1,638,528
ReLU-16	[-1, 128, 212, 212]	0
Conv2d-17	[-1, 128, 212, 212]	409,728
ReLU-18	[-1, 128, 212, 212]	0
Conv2d-19	[-1, 128, 212, 212]	147,584
ReLU-20	[-1, 128, 212, 212]	0
Conv2d-21	[-1, 128, 212, 212]	409,728
ReLU-22	[-1, 128, 212, 212]	0
Conv2d-23	[-1, 128, 212, 212]	409,728
ReLU-24	[-1, 128, 212, 212]	0
Conv2d-25	[-1, 256, 212, 212]	33,024
ReLU-26	[-1, 256, 212, 212]	0
Conv2d-27	[-1, 64, 212, 212]	802,880
ReLU-28	[-1, 64, 212, 212]	0
Conv2d-29	[-1, 1, 212, 212]	3,137

Total params: 4,402,113
Trainable params: 4,402,113
Non-trainable params: 0

Figure 11: Semantic Segmentation Baseline Model, Image was Resized to 212×212

4.1.1 Training

While training we also were saving predictions of test images to visualize how well our model performs. This model had no pooling layer so each epoch took a lot of time to compute and similarly, at first it was not performing great. But after some more epochs, it achieved near-perfect results which can be observed here.

In epoch 3 the output is shown in figure 12, then at epoch 13 the output is shown in figure 13 and finally the best results which is actually can be considered a satisfying result shown in 14

Figure 12: Baseline Model Epoch: 3

4.1.2 Loss Function

For calculating the loss of training and test datasets, we used a multi-loss approach. In this model, we calculated two losses and gave our model fifty percent weights for each of them. We used *Binary Cross Entropy with Logits* from *Pytorch* and the *dice loss*. We consider the dice loss a measure of overlap between two samples. This measure ranges from 0 to 1 where a **Dice** coefficient of 1 denotes perfect and complete overlap.

The loss of our training is shown in figure 15 and the test loss is shown in 16.

Figure 13: Baseline Model Epoch: 13

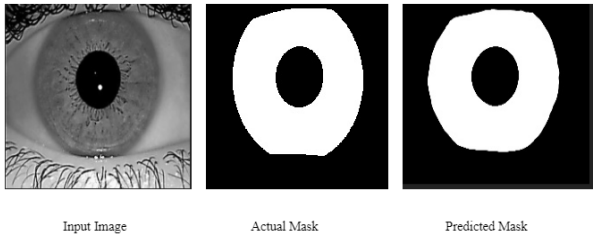

Figure 14: Baseline Model Epoch: 67, Lowest Validation Loss

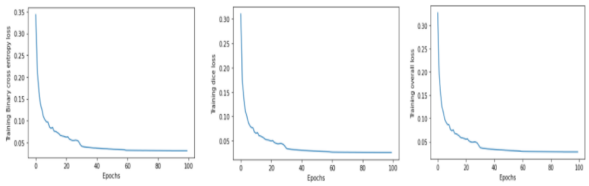

Figure 15: Training Loss

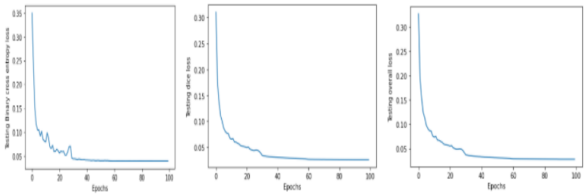

Figure 16: Testing Loss

4.1.3 Evaluation

For the segmentation accuracy evaluation, we compare the predictions obtained from each of the models trained in this study against manually annotated, ground truth binary masks. For each pair of a predicted segmentation and a ground truth mask, two accuracy metrics typically employed for accuracy assessment in segmentation tasks are used:

- Intersection Over Union (IOU)
- F1-Score: which is better for unbalanced labels like in our case as the background can shadow the bad performance for the foreground pixels.

4.1.4 Results

For our baseline model, the scores are in table 10

4.2 U-Net

UNET is a convolutional neural network architecture that has a contracting path and expansive path.

Metric	Train	Test
F1-Score	0.979	0.975
IOU	0.979	0.964

Table 10: Baseline Model Results

- Contracting Path: spatial info reduced but feature info increases.
- Expansive Path: combines features and spatial info through up-convolutions and concatenations from high-resolution features from the contracting path.

The concept behind this architecture is to reduce the height and width of the input to make a small representation of main features and then using the skip connections and that representation to output the mask such that our loss decreases. The architecture used has $5 - block$ for down-sampling and encoding and $5 - blocks$ for up-sampling and decoding

4.2.1 Training

While training we also were saving predictions of test images to visualize how well our model performs. This model performed great from the first epoch so after few epochs its performance improvements became small. It achieved near-perfect results which can be observed here. Shown in the following figures 17, 18, and 19 the progress of the model's training in epoch 1, 20, and 50 respectively.

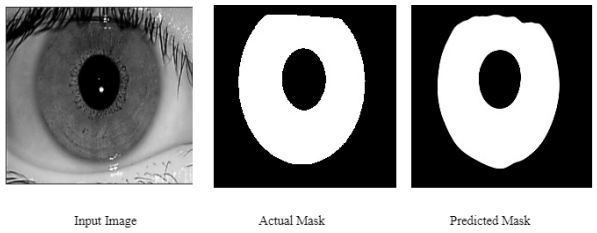

Figure 17: U-Net Training Epoch: 1

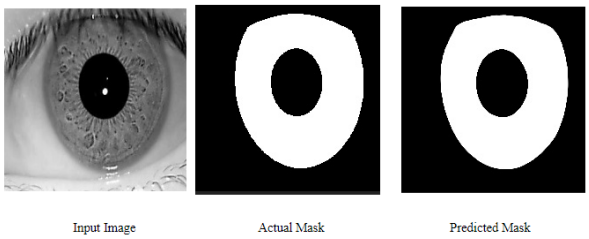

Figure 18: U-Net Training Epoch: 20

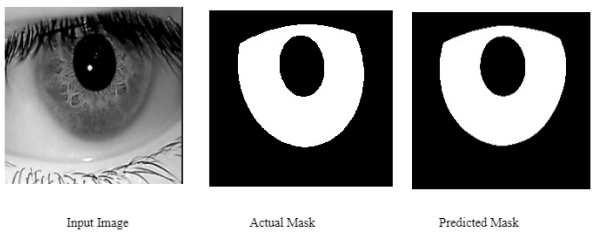

Figure 19: U-Net Training Epoch: 50

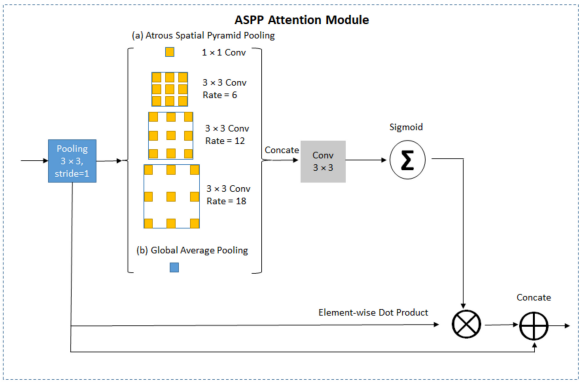

Figure 20: ASPP Attention Module

4.2.2 Loss Function

The same multi-loss function as the baseline model is used for the U-Net too.

4.2.3 Results

For the U-Net model, the scores are in table 11

Metric	Train	Test
F1-Score	0.997	0.98
IOU	1	0.98

Table 11: U-Net Results

4.3 IRU-Net, U-Net with Attention

IRU-Net [LPAC21] is a modification on U-Net. The idea is to add more contextual information to the U-Net by adding a small attention module at the smallest encoding before the first up-sample to help the model train which features are more important than other and It achieves good results that are slightly better than the U-Net. The module is shown in figure 20.

4.3.1 Results

The results are comparable to U-Net with simple addition to help the model capture more contextual information, the result are shown in table 12

Metric	Train	Test
F1-Score	0.991	0.98

Table 12: IRU-Net Results

5 Recognition

For the identification of a person from his iris image, we did some preprocessing of the input data and then tried some approaches which each yielded good results.

1. Simple multi-class classification using the iris normalized form
2. Iris Representation Learning

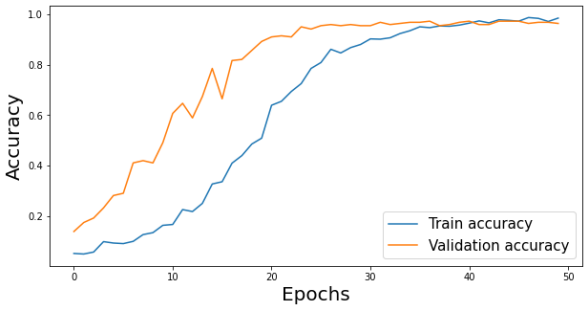

Figure 21: Training and Validation Loss

5.1 Multi-class Recognition

The model is following the paper [AWQI+18]. This model uses a simple CNN approach followed by a softmax classifier to perform extract discriminative features of the iris and obtain recognition. The preprocessing that is implemented to find the normalized iris image.

The data is then normalized and one hot encoded. Then the data is fed to CNN followed by the softmax classifier layer. We add a dropout to reduce the overfitting. We can see in the figure that the validation accuracy is higher than the training one as shown in figure 21.

5.2 Representation Learning

For the recognition we followed the *faceNet* recognition approach for our task with some modifications.

Here we tried to learn the embedding vector for each of the images to best represent them using *triplet loss function* with a simple *googlenet* architecture. This way at the end we were producing embeddings as 128 feature vectors.

5.2.1 Preprocessing

Currently, we have the access to images of the iris. So to make the problem statement easier, we multiplied the segmented mask (obtained from UNET mentioned above) with the image to remove the background and pupils. So we only pass the iris part to our model.

5.2.2 Triplet Loss Function

As shown in equation 3, Basically what it does is reduces the distance between positive image and anchor image embeddings and increases the distance between anchor image and negative image embeddings.

$$L(A, P, N) = \max(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0) \tag{3}$$

Our data generator takes a random image as an anchor image and tries to take a positive image from the same class but not the same image. Then it chooses any random negative image from the rest of the classes except that image as shown in figure 22. We didn't implement the hard positives and hard negatives as described by FaceNet. Then we multiplied them with the mask and passed them to the model.

The features vectors resulting from this model have decidability index of 3.45 which is very high compared to the other classical feature extraction methods. This is due to the fact that either triplet loss or contrastive loss are trying to create feature vectors such that the distance between the values of similar classes are small and different classes are big, also since that the *DI* depends on the absolute mean difference between the distributions of the feature vectors of each class, the output features are designed to increase the *DI*.

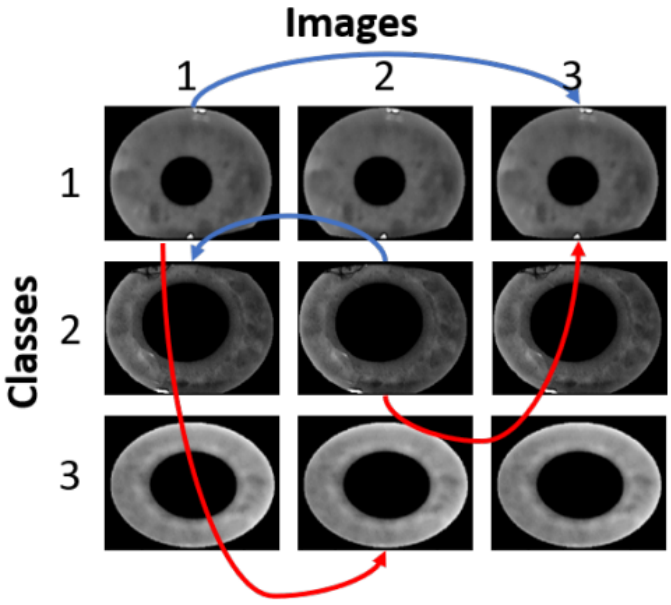

Figure 22: Triplet example, the blue lines show positive and anchor and red shows the negative image for that example of triplets

5.2.3 Contrastive Loss Function

The Contrastive loss function shwon in 4 is also used to train a representation learning model. the loss is similar to the triplet loss but it is much easier to implement and use in terms of the data generator development and GPU resources usage, since each sample in the triplet loss needs a forward pass on three different images then needs to perform the backward pass on all of them at the same time while the contrastive loss needs only a pair of images at a time.

$$y * D^2 + (1 - y) * \max(0, m - D)^2 \tag{4}$$

where D is the euclidean distance between the two feature vectors of both images and y is the label, 1 it the two vectors belongs to the same class and 0 otherwise.

5.2.4 Network Architecture

We used the GoogLeNet model for our representation learning. This model has twenty-two layers and inception modules.

- Inception module allows to implement deep (many layers) and wide (many parallel operations) nets, w/o substantially increasing the computational cost! As it computes multiple different transformations over the same input in parallel and at the end concatenates into a single output.

5.2.5 Inference and Machine Learning

The results are hard to interpret based on the value of the loss alone and the inference is also hard just based on the model. The solution is to perform machine learning algorithms. Table 13 shows the results for different machine learning techniques and *SVC* model has the highest score in recognition so far.

References

[AWQI+18] Alaa S Al-Waisy, Rami Qahwaji, Stanley Ipson, Shumoos Al-Fahdawi, and Tarek AM Nagem. A multi-biometric iris recognition system based on a deep learning approach. *Pattern Analysis and Applications*, 21(3):783–802, 2018.

Model Name	Train AUC	Train EER	Test AUC	Test EER	Best parameters
Logistic Regression	1	0.454	0.999	0.443	{'C': 50, 'penalty': 'l2'}
Desicion Tree	1	0.004	0.89	0.004	{'max_depth': 181}
Random Forest	1	0.487	0.999	0.455	{'max_depth': 11, 'n_estimators': 2500}
SVC	1	0.454	0.999	0.451	{'C': 50, 'kernel': 'sigmoid'}
GaussianNB	1	0.461	0.999	0.458	{'var_smoothing': 0.081}
KNN	0.999	0.017	0.992	0.017	{'n_neighbors': 9}

Table 13: Model's Inference Based on Classical Machine Learning

[Dau09] John Daugman. How iris recognition works. *The Essential Guide to Image Processing*, page 715–739, 2009.

[KP10] Ajay Kumar and Arun Passi. Comparison and combination of iris matchers for reliable personal authentication. *Pattern recognition*, 43(3):1016–1026, 2010.

[LPAC21] Yung-Hui Li, Wenny Ramadha Putri, Muhammad Saqlain Aslam, and Ching-Chun Chang. Robust iris segmentation algorithm in non-cooperative environments using interleaved residual u-net. *Sensors*, 21(4):1434, 2021.

[RAA+19] Humayan Kabir Rana, Md. Shafiul Azam, Mst. Rashida Akhtar, Julian M.w. Quinn, and Mohammad Ali Moni. A fast iris recognition system through optimum feature extraction. *PeerJ Computer Science*, 5, 2019.