Iris recognition system

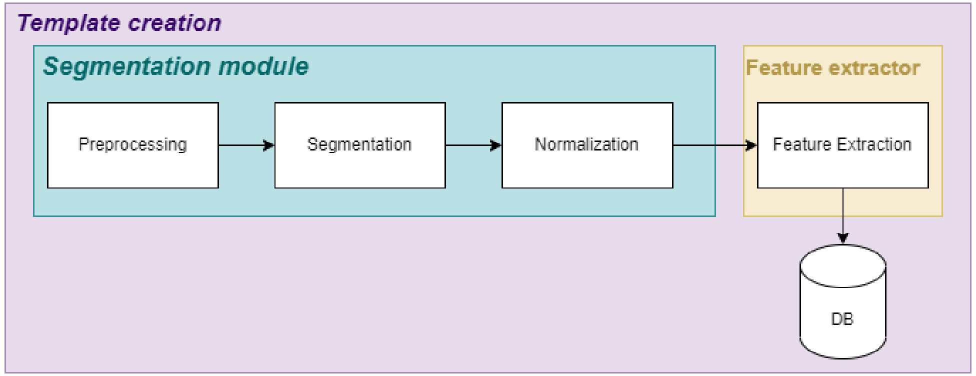
DL based iris feature extractor trained with few-shot learning

Eduardo Rinaldi - 1797800

Introduction

- For my bachelor's thesis I decided to work on an iris recognition system which relies on $IS_{IS}v.2$ method for segmentation and both *LBP* and *Spatiogram* operators for feature extraction
- For this project, instead, I decided to circle back and **try other approaches** regarding iris recognition (for both segmentation and feature extraction)

Idea



The pipeline is the “*classical*” one, I just subdivided it in easy to change submodules:

1. **Segmentation** → “IS_{IS} *v.2*” & “*Hough approach*”
2. **Feature extraction** → Deep learning based approach

Technologies

OpenCV

G. Colab

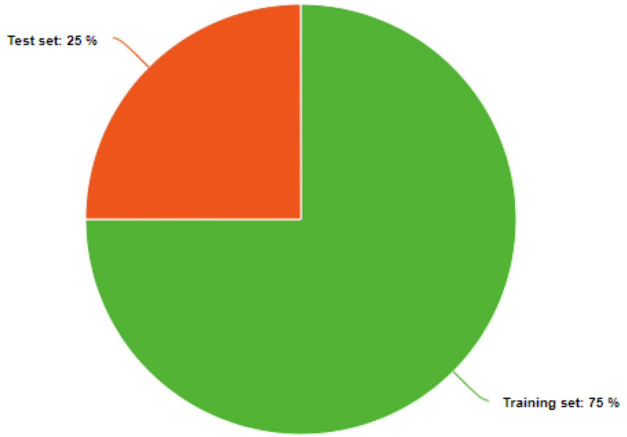
Pytorch

Pandas

Dataset

Dataset used is **Utiris**:

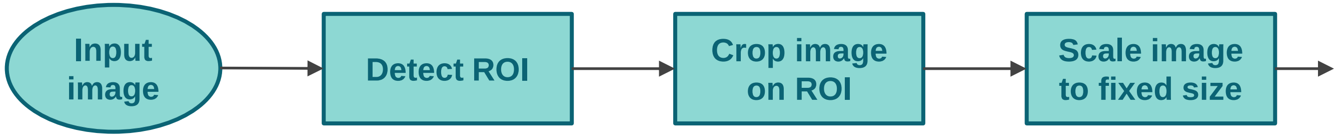
- High quality images captured in a controlled environment with both VW and NIR
- 1540 images from 79 individuals from **both right and left eyes** demonstrated in 158 classes in total
- Train/Test split **by subjects**, not by probe



Segmentation modules

- Two segmentation modules implemented
  - IS_{IS} v.2
  - *“Hough approach”*, this is the one **used for the “final” pipeline**
- Code structured so that new modules can be easily added
  - A segmentation module can be seen as a function that takes an image focused on an eye and returns two circles (one for limbus and one for pupil)
  - In each segmentation module we’re “describing” how to find these two circles
- They share a common part which consists in:
  - **Preprocessing:** speeds up computation time by removing useless ” to-compute” pixels and removes reflections
  - **Normalization:** iris is “unrolled” in a rectangular image

Segmentation modules - Preprocessing



Using an **Haar Cascade Classifier**

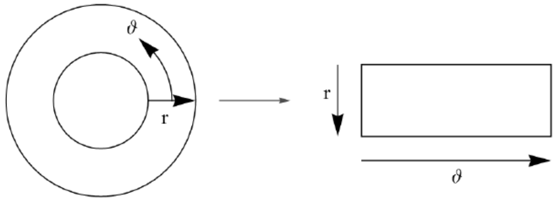
Segmentation modules - Normalization

Normalising with the **Homogeneous Rubber Sheet Model**, which transforms an image *I(x, y)* in *cartesian coordinates* into one expressed in *polar coordinates* as follows:

$$I\big(x(r,\theta),y(r,\theta)\big) \rightarrow I'(r,\theta)$$

$$x(r,\theta) = (1-r)x_p(\theta) + r_{xL}(\theta)$$

$$y(r,\theta) = (1-r)y_p(\theta) + r_{yL}(\theta)$$



Rubber Sheet Model

Example of normalized iris

IS_{IS} v.2

- Based on **“bruteforce” approach** (for both limbus and pupil): generates all the possible circles using different parameters and then select the best one according to a specific metric
- Two main phases:
  - **Circle candidates generation:** posterization filter → canny edge detection → taubin circle fitting
  - **Circle selection:** receives as input a list containing all the candidates circles obtained after each iteration of the previous step and returns only one circle based on some constraints and criteria

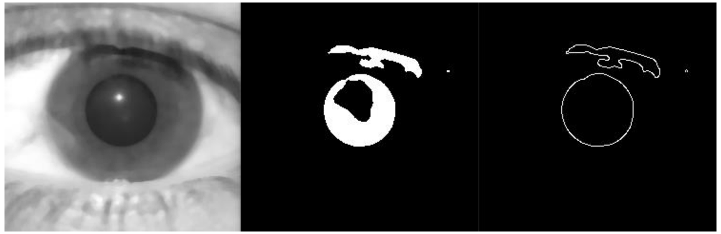
Hough approach

- Custom approach created by mixing *“IS_{IS}”* with some of the approaches used here even though this latter is thought to work well on **infrared images**
- The main idea here is to find **first the pupil** and **then the limbus** still using a “bruteforce” approach based on “*circle candidates generation*” and “*circle selection*” steps

Hough approach - Circles generation

This step is **repeated several times**, and is composed by the following substeps:

1. Input image is converted into **grayscale**
2. **Median blur filter** is applied to the grayscale image (each iteration with a different window size), obtaining a blurred image
3. As in "IS_{IS}" , **canny edge detector** is used for obtaining circle edges and it receives as input
   a. for limbus, the blurred image
   b. for pupil, since pupil circle is composed by very **dark pixels**, I can **inverse threshold** the blurred image with small threshold values (at each iteration this threshold is increased) for then giving the resulted image to the canny edge detector; this latter in this case will have more chances to detect edges of pupil
4. Final step consists in giving the "edge map" (i.e., the output of canny) as input to ***"Hough circle transform"*** which will give as output a list of all possible circles



From left to right: blurred image, thresholded image, edge detected by canny

Hough approach - Circle selection

This step takes as input the list of all circles found in the previous step (after all the iterations), and:

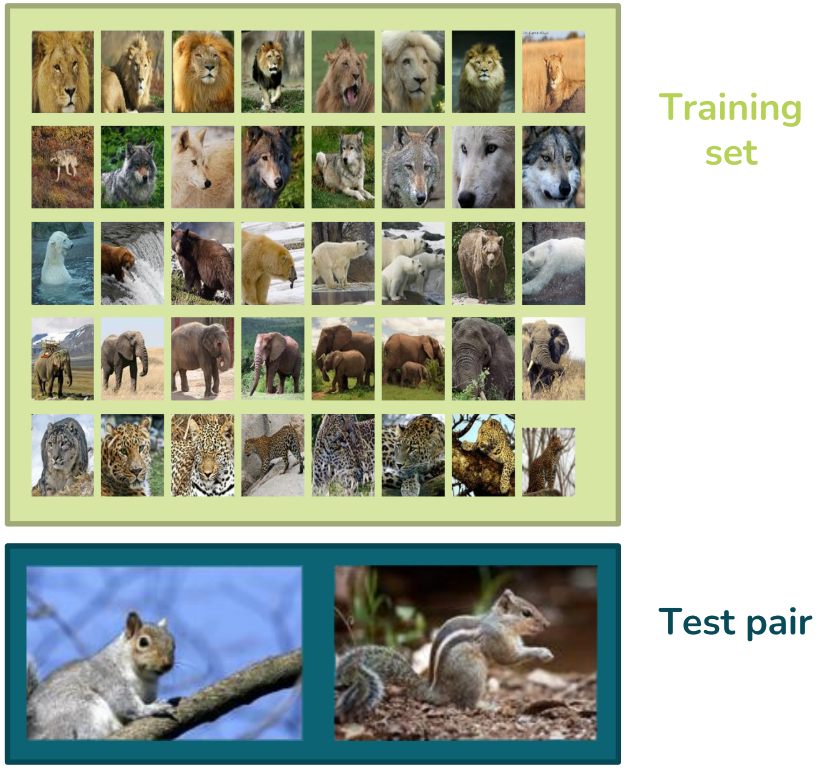
- For pupil is very simple: just take the mean circle (i.e., mean position and mean radius)
- For limbus, a filtering process is applied:
    - discard every circle with $r_L < 1.5\ r_P$
    - every circle positioned outside pupil circle is discarded
    - mean circle μ and std circle σ are calculated, then every circle with position or radius outside the range $[\mu\ - 1.5\sigma,\ \mu\ +\ 1.5\sigma]$ is discarded
- At the end of this filtering process, as for pupil, the new mean circle among the candidates is taken.

Feature extraction - Preface

- Initially, the idea was to use pretrained networks (*vgg*, *densenet*, ...) and then perform a fine-tuning operation for the proposed dataset
- This idea was immediately catalogued as a **failure** for 2 main reasons:
    - for each class there are very few examples (about 5), not enough even for a small fine-tuning process.
    - if a new subject is added to gallery, we need to retrain the whole model.
- So, I came up with a new idea: use ***"few-shot learning"*** and ***"siamese networks"***

Few-shot learning

- Few-shot learning is the problem of making predictions based on a limited number of samples.
- The goal here is not to let the model recognize the images in the training set and then generalize to the test set, instead, the goal is to learn ("*learn to learn*")
- Learn a **similarity/distance** measure



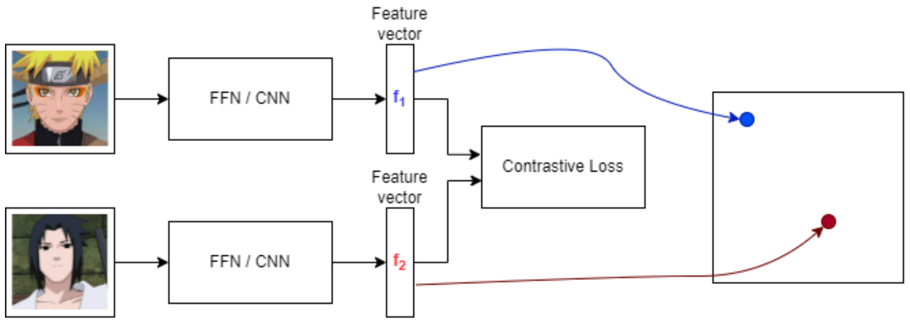
Siamese networks

- A possible implementation of few-shot learning is through the concept of Siamese networks
- A Siamese network consists of **twin networks** (generally two or three) which accept distinct inputs but are joined by an energy function and they **share the same weights**
- **Note:** two extremely similar images are not mapped by each network to very different locations in feature space because each network computes the same function

Pair siamese networks

- While training, an **image pair** is fed into the model with their ground truth relationship: y: $y = 1$ if the two images are similar (in our case from same iris) and 0 otherwise
- The loss function for a single pair is:

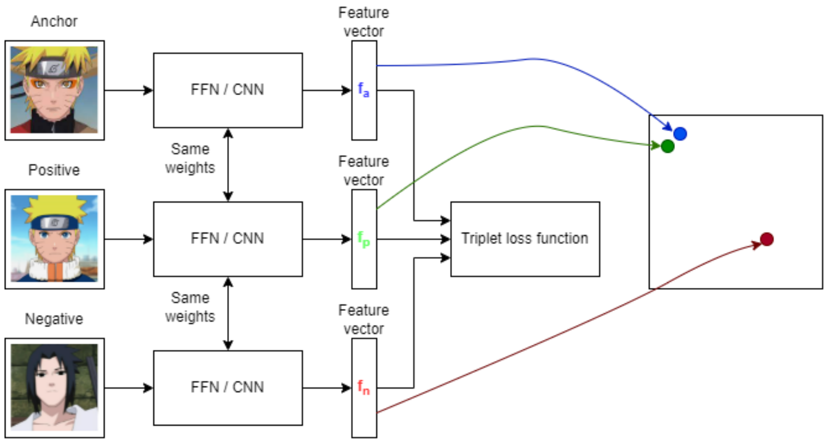
$$ContrastiveLoss = yd^2 + (1-y)\max(m-d,0)^2$$

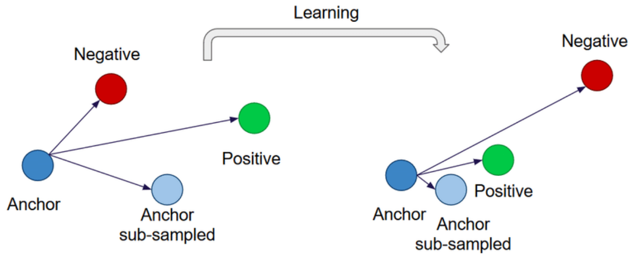


Triplet siamese networks

- During training process, an **image triplet** $< I_a, I_p, I_n >$ is fed into the model as a single sample
- I_a, I_p and I_n represent the **anchor**, **positive** and **negative** images respectively
- Distance between anchor and positive images should be smaller than between anchor and negative images; training process optimize this condition.
- The loss function used is $TripletLoss = max(\left\|f_a - f_p\right\|^2 - \|f_a - f_n\|^2 + m, 0)$

Triplet siamese networks





Trained and tested models

- **vggFEPretrained**: pretrained model of "*vgg*" as it is, no fine-tuning applied, used as feature extractor as it is
- **vggFEPair**: pretrained model of "*vgg*" fine-tuned using pair few-shot learning (pair siamese network)
- **vggFETriplet**: pretrained model of "*vgg*" fine-tuned using triplet few-shot learning (triplet siamese network)
- **featNetTriplet**: inspired by "*featNet*" architecture and trained from scratch (weights initialized randomly)

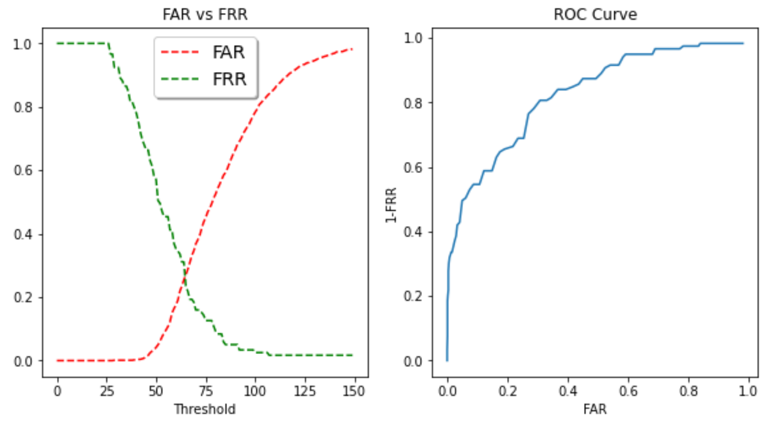
All trained for **100 epochs**

Evaluation

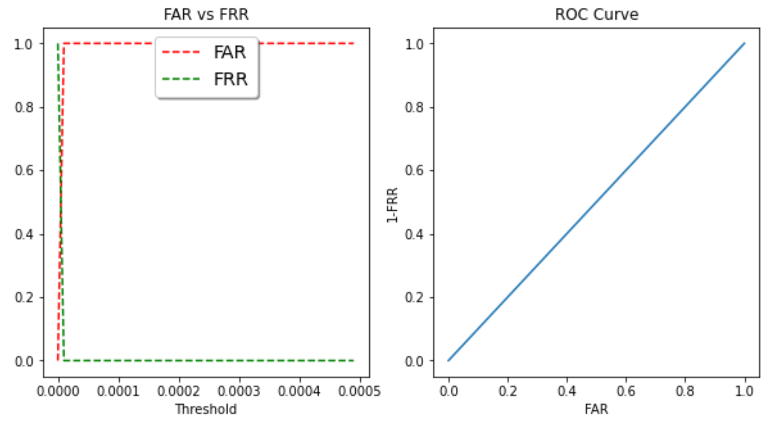
For the evaluation phase I measured the performance of the different models on:

- **All-vs-all verification** multiple template, obtaining 3 main informations: *FAR*, *FRR* and *EER*
- **All-vs-all identification** open-set multiple template, obtaining the following main informations: *FAR*, *FRR*, *EER*, *DIR* (Detection and Identication Rate) and *CMS* (Cumulative Match Score)
- Then I calculated the **distance matrix** between all samples in the test set, using the **Euclidean distance** (distance values are not normalized).
- Immediately after that I used the algorithms seen in the lecture for the verification phase and the identification phase, using not the eye id but the subject id as the label

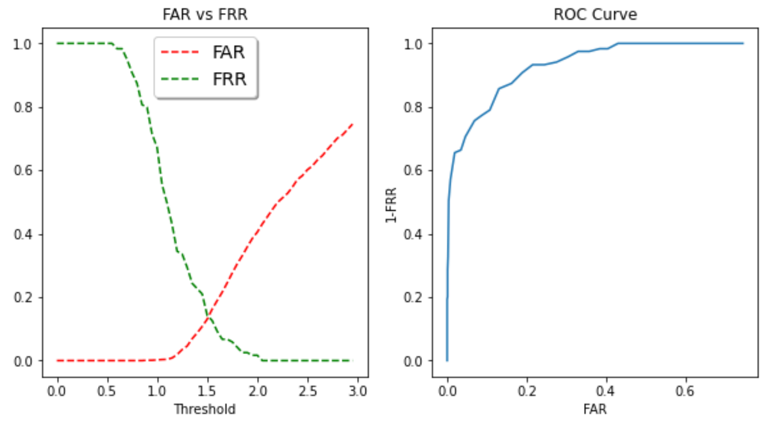
Verification results



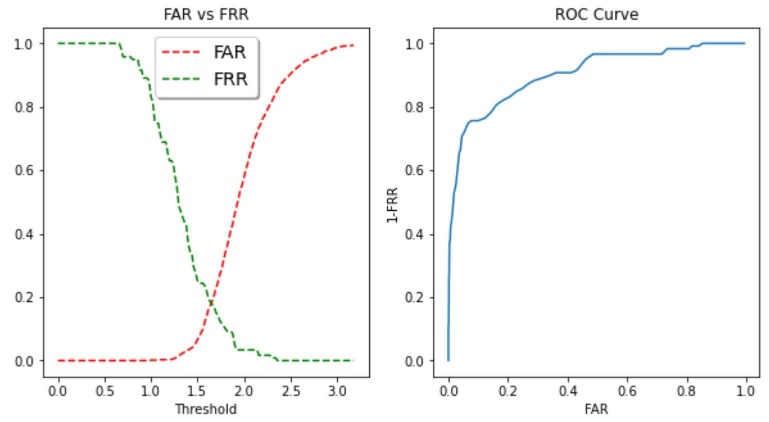
vggFEPretrained



vggFEPair

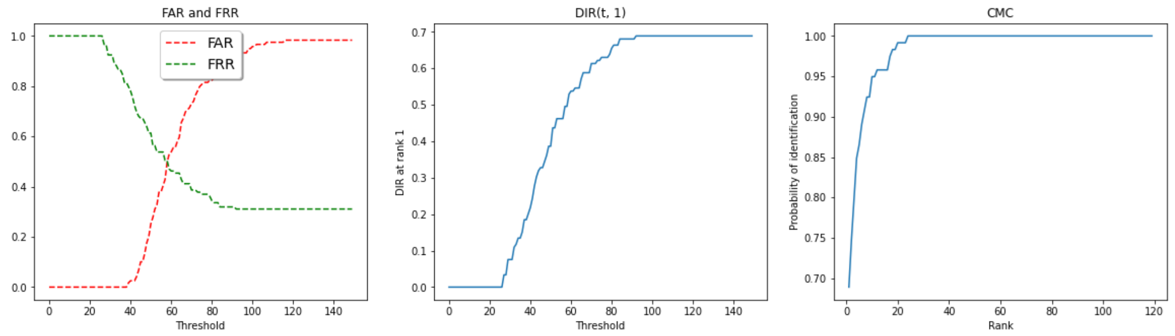


vggFETriplet

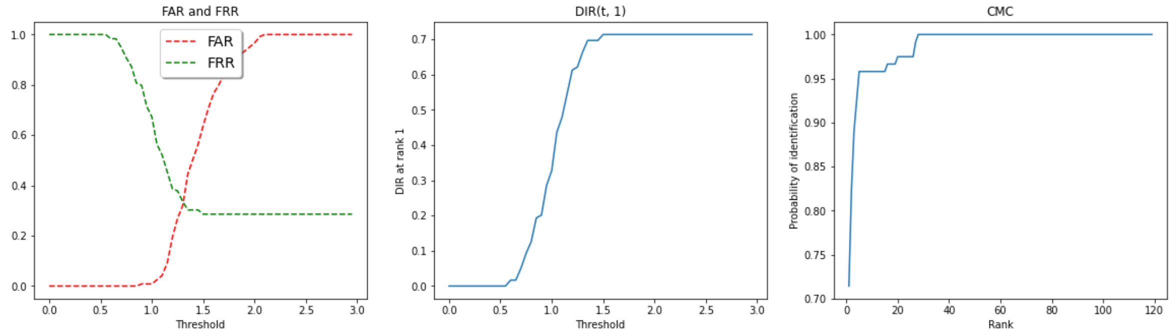


featNetTriplet

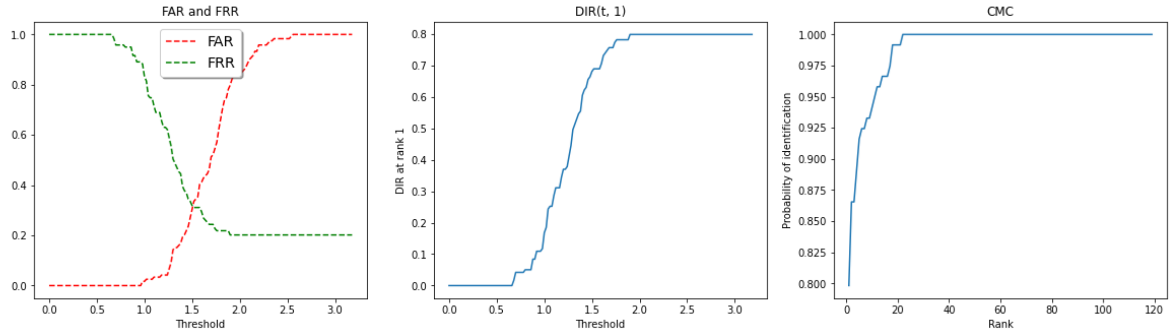
Identification results



vggFEPretrained



vggFETriplet



featNetTriplet

Final considerations

- “*vggFEPretrained*” compared to others does not perform so bad on verification ($EER =\sim 0.25$).
- The **best model for a verification** task is “*vggFETriplet*”, since the $EER =\sim 0.16$. Note that a valid second choice could be “*featNetTriplet*” with $EER =\sim 0.2$
- The **best model for an identification** task is “*featNetTriplet*”, with:
  - $EER =\sim 0.3$ at $t = 1.5$
  - $DIR(t = 1.5, 1) \ = \sim 0.7$
  - $CMS(rank \ = \ 1) \ = \ 0.8$
- **Triplet training** performs way **better than pair training**

Demo

The demo consists of 3 python scripts on CLI:

1. **"Enrollment.py"** to perform enrollment of a subject
2. **"Verification.py"** to perform a verification operation by specifying the input image and the claimed identity
3. **"Identification.py"** to perform an identification operation by specifying the input image

For simplicity in the demo, I decided to use only *“featNetTriplet” model as feature extractor.*

Future works

Since time was limited for me, I had to choose a subset of things to do; here I list several things that could be interesting to test:

- Train and test all 4 feature extractors on a mixed dataset (e.g., *Ubiris* + *Utiris* + *MICHE*)
- Plug different feature extractors: could be a different pretrained network (e.g., “densenet”), or a custom network to train from scratch
- Plug a different iris segmentation module

Thanks for your attention!