Biometrics - Report assignment 2

Fingerprint and Iris based identification

Snyers Harold - r0880356

Professors : Peter Claes

April 29, 2022

Contents

Fingerprint and iris based identification

In this report we will focus on implementing and testing a keypoint based fingerprint and iris recognition system. We will look at local similarity, global similarity and a hybrid version of those two systems, and a fusion of these two systems as well. Finally, we will also look at a iris recognition neural network.

1 Context and Data

We will evaluate these systems through a murder case. A woman is found dead in her hotel room and although we know that 15 minutes before her death someone entered in her room alongside her, the murderer was very careful to not disclose anything that would identify him/her. Fortunately, our team found a fingerprint and was also able to extract the iris of that murderer. Our team has assembled a database of all the fingerprints and irises of the persons that were present in the hotel and our goal is to be able to solve the case by implementing recognition system based on iris or fingerprint.

As we know, the fingerprint and the iris are both unique physiological characteristics of the human and can thus be used for forensic. When working with fingerprint, we analyze the pattern of the ridges and valleys on the surface of the fingertip. Most often, the right index or thumb is used. For the iris, we look at the pattern in the texture of the iris. The fingerprint database is made out of 100 gray images of size 480x320. The iris database is made out of 100 gray images of size 280x320. Sample images can be found in the notebook.

2 Implementation fingerprint recognition system

Now that we have looked at the data, we can proceed for the analysis for the best similarity function for the fingerprint and iris, and even for a combination of boths. This analysis will be done by through a series of questions that we will answer sequentially in order to evolve the best solution.

2.1 Simple Similarity function

Q1: Now that we have some results, is the given similarity function a good metric to quantify the distance between two fingerprints? Is it reliable enough to incriminate a suspect? What are its limitations?

Using the euclidean distance as a distance metric alone on the raw images will most likely not give any good result. This is because we are trying to score the similarity of the different fingerprints based on the acquisition data. The captured fingerprint will most likely have discontinuous ridges due to poor resolution, acquisition, or other factors which can affect the matching, as well as the surrounding of the background of the image. However, this gives us a first opportunity to compare different distance metrics that can used for the similarity score.

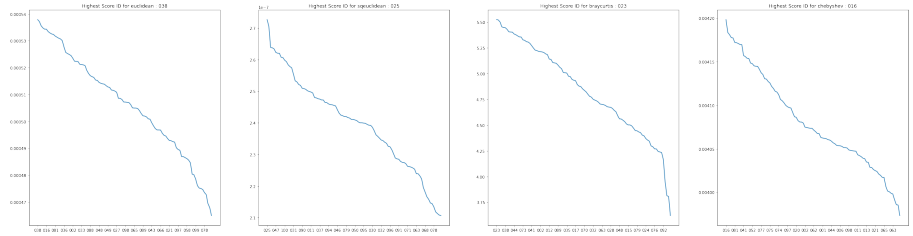


Figure 1.2.1: Comparison of 'euclidean', 'sqeuclidean', 'braycurtis' and 'chebyshev' pairwise distance function from the scikit-learn library

Consequently, we can definitely conclude that this similarity function is not reliable at all for forensic purposes. This is clearly showcased in figure 1.2.1 where we have no clear pattern that suggest that one person's fingerprint is more similar the the perpetrator fingerprint.

2.2 Model Extension : Local similarity

We can now proceed to first apply some preprocessing to our fingerprint database in order to have a segmented and enhanced image of the fingerprint. This important step is heavily based on the work of Ukarsh Deshmukh and Peter Kovesi. The enhancement is done with a Gabor filterbank using gradient-based orientation and local frequency estimation. For the segmentation of the fingerprint from the background, a mask is computed by calculating the standard deviation in local windows and thresholding above a certain level.

2.2.1 Keypoint detection and feature descriptor

For the local similarity, we will first need detect certain keypoints on the fingerprint. Several keypoint detectors and descriptors exist. We will try to briefly analyse 5 of them; 1) ORB, 2) SIFT, 3) Harris, 4) FAST, 5) BRIEF (or Star).

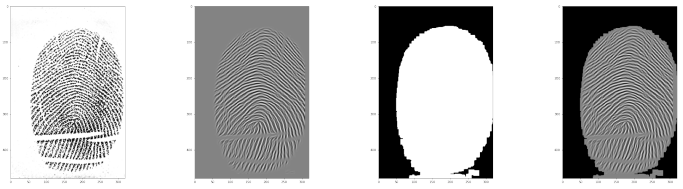


Figure 1.2.2: Comparison of raw (1), enhanced (2), masked (3) and segmented (4) image sample

Assignment 1 : [FING and IRIS] OpenCV provides different KeyPoint detectors and descriptors (ORB, SIFT, SURF, BRIEF, . . .). Briefly test, visually, which of these seem to extract relatively reliable interesting points from the fingerprints or iris dataset (you can skip the ones that require a lisence). Use at least 3 detectors. (1pt)

The HARRIS detector is a detector specialised in detecting corners and is rational invariant. It basically finds these corners by computing the intensity change in all directions. This detector can consequently also be applied to fingerprints because of the change in intensity of the ridges and valleys, but also works for iris prints. The SIFT detector is another detector that like the HARRIS detector is able to find corners and is rotational-invariant but on top of that is is also scale invariant. In our case this not necessarly needed, but a corner at different scales might indeed not be detected as corner all the time, e.g. as the image size grows, the corner becomes a line or curve [1].

Another detector, called FAST, was initially developed for real time applications. It works at the pixel level and finds corners by looking at the pixels inside a circle from the pixel of interest. Again, it looks at the change of intensity to determine if there is a corner or not. BRIEF or (STAR) is detector initially developed in response the memory space the other detectors need for their description vector. With a certain shortcut, it is able to find locations of interest such as corners without finding descriptors. Finally, the ORB detector is an algorithm developed by opencv in response the patented algorithms such as SURF and SIFT. ORB stands for oriented fast and rotated brief and is thus basically a fusion between the fast and brief detector [1].

As illustrated in figure 1.2.3, the ORB and FAST detectors have much bigger range of detected keypoints. However, this doesn't mean that the other detectors are bad although a higher number of keypoints would make the comparison more reliable. However, having a higher number of keypoints will give more probability to have false positives.

Detectors such are ORB and SIFT have the capabilities to both detect keypoints and compute their description. The other make use of descriptor to compute the

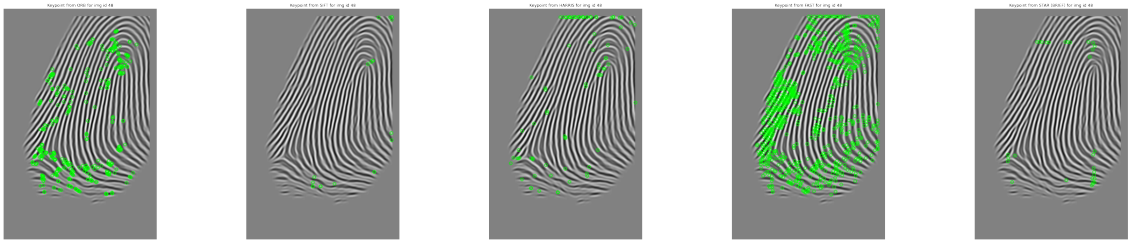


Figure 1.2.3: Comparison of ORB, SIFT, HARRIS, FAST and STAR detectors for sample id 48.

description from their detected keypoints. In this case, we use the BRIEF keypoint descriptor. An example is shown in the image below where we illustrate the local descriptor vector of the ORB detector. Except for the SIFT detector which uses a 128 dimension vector of its descriptor, they all compute a vector of similar length, namely 32 dimension.

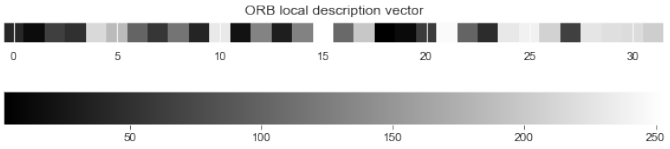


Figure 1.2.4: ORB local description vector for sample 48

Finally, to determine if the keypoints of one fingerprint matches the other, one can try to find the matches by bruteforce and return the set of distance between these matched pairs. With the ORB detector, we use hamming distance to compute the distances while for the others we use the L2 distance for compatibility issues for the hamming distance.

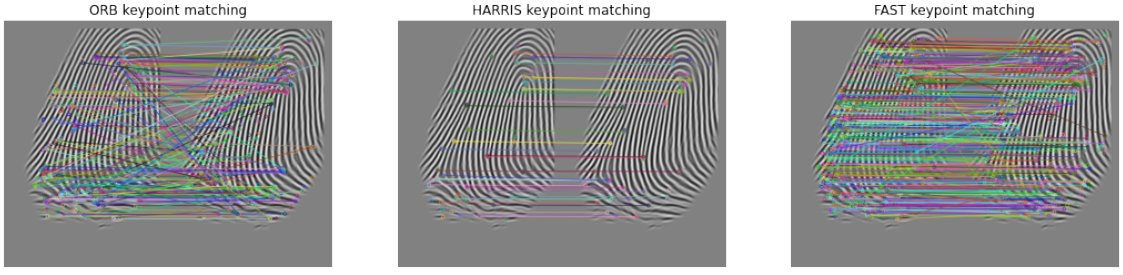


Figure 1.2.5: Comparison of matching keypoints between ORB, HARRIS and FAST

Q2: Are all the keypoint matches accurate? Are they expected to be? Explain why.

The HARRIS detector with the brief descriptor seem to have the best matching result. Although the FAST detector has many keypoints, the matching seems to be much more accurate than the ORB detector. As the two images are very similar or the same, the matching are expected to be accurate.

After having brute forced the search for these matches, we simply apply a simple scalar measure on the set of distances of these matches. Different solutions can be used here such as counting the number of matches that have a distance smaller than a certain threshold. Depending on the detector, we needed to define a different threshold but overall, a threshold of 50 gave good results for all five detectors. The ORB detector was the only one that gave better result with a smaller threshold. Another method that doesn't require us to test out which is the best threshold for that similarity measure is apply a scalar measure on the N smallest distances (here we choose 20). As scalar measures, we implemented the mean, the sum and the harmonic mean. All these scalar measures, eventually give a very similar result. Another advantage to this last method is that we get a non zero score for almost every compared image while still getting a clear distinction between matching pairs.

Comparing the detectors based on the scalar measures, we can see that the HARRIS, FAST, and ORB detector show a greater difference between matched pairs and unmatched pairs (unmatched pairs scores are more flat).

2.3 Model Extension : Global similarity

Another approach for computing the similarity scores is to use global features. These global features are computed by first using the local matches to align the the fingerprint, basically finding the the transformation needed to align the compared pairs. Once aligned, we can directly compare the keypoints from a geometrical view. The alignment step was already provided.

We also found that thanks to this alignment, the matches between two images were much better, even for the ORB detector where all drawn matches are now horizontal and thus corresponding as expected.

Q3: Choose a global feature similarity function, (e.g. you can start from euclidean distance between the reduced sets of KeyPoints and count the values above a threshold).

Again, after computing the local matches and the global matches from the latter, we need to define an image similarity distance to compute scores from the global features. As shown in the code below, we first extract the keypoints from both images and for each match we apply a certain distance function for both keypoints. Then we loop through all those scores and keep only the scores above a certain

threshold defined for each of these distance functions. The final score is the length of those remaining distance scores.

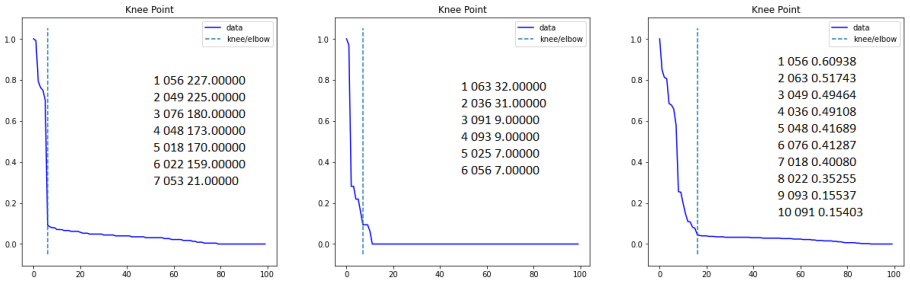
```
def global_img_similarity(matches, reg_kp1, kp2, dist_fct_n,
    thresholds):
    kp1_reduced,kp2_reduced = get_reduced_set(matches, reg_kp1, kp2
    )
    best_scores = []
    for i, match in enumerate(matches):
        score = distance_sklearn(kp1_reduced[i].reshape(-1, 1),
    kp2_reduced[i].reshape(-1, 1), dist_fct_n)
        if score < thresholds[dist_fct_n]:
            best_scores.append(score)
    best_scores = np.array(best_scores)

    return len(best_scores)
```

Q4: Visualize the scores and determine a score threshold to discriminate the matching fingerprints. Explain how you determine the threshold.

The Scikit-learn library provides a whole set of pairwise distance function. We have compared eight of them ('sqeuclidean', 'braycurtis', 'canberra', 'chebyshev', 'cityblock', 'euclidean', 'l1', 'l2', 'manhattan') with the different detectors to find out which pair of metric and detector worked best. For each distance function, a threshold needed to be determined. This threshold was chosen as the value that would make the similar images distinct from the non similar while keeping it high enough to have enough scores left for a clear comparison.

We first found that the SIFT detector, while always outputting the person with id 56 as first, gave different results than the other detectors for all distance function with false similar fingerprints. The FAST detector returned robust results where we could always clearly see the 6 "common images" in the top 6. On top of that, the distinction from the rest was very clear. The distance function didn't influence the results all that much except for order of the ids, but showed in all cases the same curve shape and same top id, 56.

For the ORB detector, although we are able to see a knee and thus distinction between the similar and non similar fingerprints, the curve was less clear for a few of the distance functions. As with all the other detectors, the sqeuclidean, braycurtis, and canberra distance function showed better results than the other distance functions. Again, the person with id 56 was mostly in first place as well. For the HARRIS detector, we were able to see also a clear distinction between similar and non similar fingerprints. Interestingly, the top 6 order was different than ORB's or FAST's with person id 48 and 76 always in the top 2. Finally, the STAR detector even worse result that the SIFT detector with only false similar fingerprints. Here under is an example of the



(a) Fingerprint similarity (b) Iris Similarity (c) Multimodel similarity

Figure 1.2.6: Comparison different systems (a) Finger print global knee point for FAST detector with sqeuclidean distance function (b) Iris hyrbid similarity function knee plot for SIFT detector with braycurtis distance (c) Iris and fingerprint similarity Similarity function knee plot for SIFT detector with braycurtis distance for the iris and FAST detector with sqeuclidean for the fingerprint

2.4 Model Extension : Hybrid similarity

Q5: Choose a hybrid feature similarity function that makes use of both the geometric distance and the feature distance of the keypoints. Using this hybrid function, visualize and assess the matches.

The hybrid similarity function uses the same code as the global similarity except that it uses the also the match distance as an additional threshold to filter out the resulting scores. With this additional threshold, all the detectors, except for the STAR detector, provided correct results. In addition, the distinction between similar and non similar fingerprints was for each detector also very clear, even more than the curve shown in figure 1.6(a).

As we have concluded in the notebook, 6 fingerprints were altered such that they are all similar. Throughout the different comparison, local, global or hybrid, the same person id came always on top, namely 56. However we can't know for sure which is why we will analyze the iris print first to try to provide more evidence to our conclusion.

3 Implementation iris recognition system

When we use the iris as a biometrics, we are analyzing the patterns in the texture to determine one's identity. As we are looking at the texture, the images can be in black and white.

Q6: Check out iris_perpetrator.png. Where do you see difficulties? What kind of similarity measures do you expect to work best?

Looking at the perpetrator iris scan, we can see that first difficulty will be extract the iris part of the image to do a comparison. Indeed, we need to be able to extract the flat donut that is the iris. In a perfect world where the iris is alone and clear color change from the pupils to the iris and from the iris to the sclera. So the first step will be to detect where the iris is and the second step is the segmentation of the iris to leave out only the actually iris and remove overboarding eyelids for example.

Once the iris has been detected, enhanced, and segmented, we can start feature extraction and finish up with matching. All the detectors seem to extract useful keypoints although we can clearly see much more keypoints being detected with the sift and fast detectors. The star detector detected the least keypoints but from an eye point of view, all keypoints seemed to have matched correctly.

3.1 Determine best similarity table for iris

Q7: Construct a similarity table with the iris images. Use local OR global matches in order to get scores. Use the scores you get to determine the perpetrator.

We have already discussed earlier on different similarity approaches to compute a similarity table. From the trials, the best combination was to use the global or hybrid similarity with as detector the SIFT detector and distance function braycurtis. For the hybrid similarity fct, we needed to modify the threshold for the matches from 55 to 100 (because of the sift detector). As threshold for the braycurtis distance function we have 4. The result is shown in figure 1.6(b).

4 Multimodel system

Q8: Fuse your iris and fingerprint biometric system on the score level to solve the murder case! Do you feel confident in your prediction? How do you fuse the scores? Why?

We have seen until now that that perpetrator was able to both alter the fingerprint and iris database. Fortunately, there is one last system that we can develop to be able to identify the murderer. We can construct a similarity table by fusing the results of both the fingerprint similarity table and the iris similarity table.

We will use the best parameters we have found so far for both system to construct this multimodel similarity table. Before we can fuse these results, we first need to normalize both tables. One can also decide to put more weight for a certain model over another if one system seems more reliable in its results. As a final verdict, as illustrated in figure 1.6(c), person number 56 came on top with 60% similarity which is 10% more than the second similarity score. With this bigger difference between

the two first scores, we are more confident with the result of this last model although there is still some uncertainty behind it.

5 An iris recognition system with deep learning

In this section, we will briefly go over the implementation of neural network using transfer learning for recognizing iris prints. Dataset of iris are for now rather scares because of the only recent patent for that technology being lifted. That said, the CASIA iris database exists and contains different training datasets of different categories such as close-up images with size 320x280 or wider images size 640x480. Due to a lack of time, we proceeded with the smaller images for which the previous iris enhancement and segmentation model was already developed.

5.1 Preparing the data

The first step for building our model is to prepare the data for the training. Our dataset contains the left and right iris of 249 people. As both irises of the person are different, we first need to create a new folder that is going to contain each person iris a folder of corresponding iris scans. For better training, we first clean up the folders by removing empty folders or folders containing less than 4 images as we need at least 1 image for the validation set and for the test set for each category. The python package splitfolders quickly creates 3 folders based on those fixed criteria.

We can now read the images from the data set and return their corresponding numpy array, class label and id. We can then compute the enhancement and segmentation of the iris scans with previously implemented methods. We do this for the training, validation and test set. Here is a sample of their segmented images. For better training, we augment the training data to get more variations in our training set.

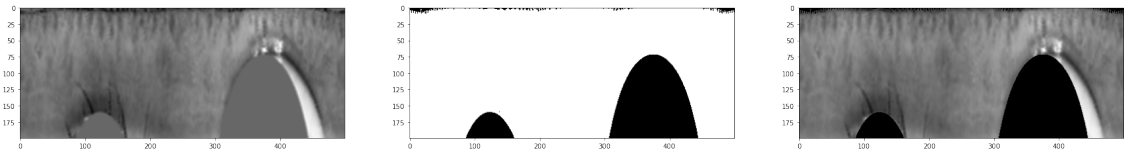


Figure 1.5.7: Example enhanced, masked and segmented image from training data set

As our training set is rather small, training on our data alone will most likely result in bad accuracy and/or overfitting. To remedy that, a popular approach is transfer learning. Transfer learning consists of using the learned features from a well known model usually on a bigger dataset and use that base model to learn the new data. We proceeded as follows, first we loaded the trained model based on the dataset

imagenet and we keep the model until the last layer before the fully-connected layers. That way, we can add our own fully connected layer and categorical layer to train on our data. For the first part of the training, we freeze all the layers of the base model. Once, we have fitted the model on our data for a few epochs, we unfreeze some of the last layers, recompile our model and fit our model again on our data fine tuning the base model.

As base model we have the inception_resnet version2, which is fusion of two popular models in computer vision. It has a total of 780 layers. For computational power purposes, the training was on a subset of the data that we have prepared. Indeed, we only kept 50 persons to simulate the training and see how well it the model could learn the new data. In below graphs, we illustrate the model training accuracy and loss on the transfer model and then afterwards on the fine tuning. As illustrated in below graph, our model was able to learn the new data rather well. However at the fine tuning we saw that the model was not able to improve upon its accuracy. Finally, we computed also the Area Under the Curve score for the receiver operating characteristic and scored only 0.5 on the test data. With further fine tuning and training, we are sure the model can improve upon that score.

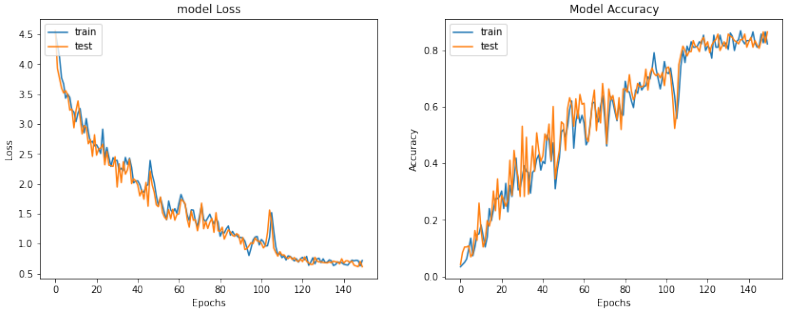


Figure 1.5.8: Loss and accuracy from training on freezed model

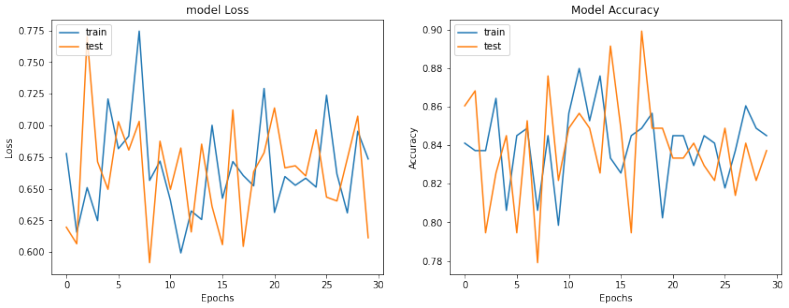


Figure 1.5.9: Loss and accuracy from fine tuning on the transfer model after unfreezing unfreezing the 80 last layers

6 Conclusion

To conclude, we have been able to implement a fingerprint and iris model our raw images. Although not totally conclusive, we have been thanks to the fusion of these two system similarity matrices to determine with high probability who the murder is.

For the different similarity measures, we have found that the local similarity in general was worse than the two other approaches. On the detector level, we can definitely say that the STAR (or BRIEF) detector was worse than the others. We found that in general the FAST detector gave the best results although ORB and HARRIS had interesting results too. For the IRIS, the SIFT detector showed in most cases better results.

Finally, we implemented a neural network for iris recognition based on the transfer learning using the inception_resnet version2 model.

6.1 Supplementary material

As supplementary material to this report, one can find a the source code made of the jupyter notebook showcasing the different implementations. For better readability and use, all the functions used in the notebook were moved to the 'src' folder under 'image_processing.py', 'matrix_constuction.py' and 'plotting_functions.py'. The source code for the neural network can be found in the submodule neural network. It contains two files each implementing a class for better usability of the model.

Bibliography

[1] Opencv (2022). Feature detection.