Final Project – Summary Report

Face Entropy

Submitters: Roy Zohar, Roy Mezan

Prof. Rita Osadchy, Dr. Orr Dunkelman

# The Problem

Biometric recognition is a fast-growing field, that has gained more and more popularity in recent years. The problem that we chose to tackle in this project is proving that despite common belief, **biometric systems aren't as safe as they seem.**

More specifically, our goal was to find exactly how "detailed" are todays' representation of faces. For example, we can all agree that a 4-digit passcode contains 10000 possibilities. However, when transitioning from a 4-digit passcode to facial recognition, this question becomes much more difficult to answer. If we are to show that facial representation contains a considerably small amount of details, we have proved that the security of facial databases is compromised.

# Our Solution

Our solution was to develop a simple "brute-force" attack on a biometric database. We accomplish this by synthesizing a large amount of artificial feature vectors, based on a set of predefined faces.

Our plan was to find 2 datasets of faces, train our model by one database, and test against the other. Synthesizing faces was achieved by sampling features from the training database.

Sampling features can be broken down into 2 steps. First off, we remove the correlations between features using PCA, since we want to sample each feature independently from the rest. This stage is necessary since there are strong correlations between features. For example, if people with large eyes often have a large nose, sampling a face with large eyes and a small nose isn't accurate. Now that we have independent features, we use the "inverse sampling" method to sample each feature independently.

# Solution Overview

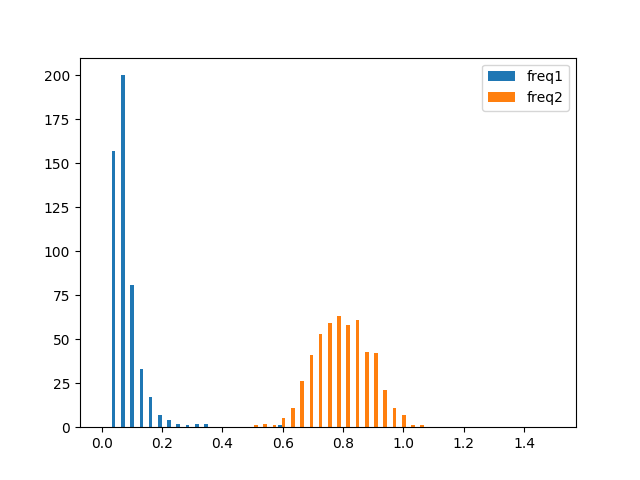
We provide a system that can find the entropy of any given database and representation method. The system runs a brute force attack on the database it is given and outputs statistics regarding the attack.

We implemented the attack using Python. We chose the DLib representation for facial features, which computes a 128-D sized vector that represents features of a face.

We used the Haifa University face databases, which contains images of 508 subjects. The reason we chose this database, was because it is consistent (lighting conditions, camera angles…). This helps our synthesizing model "concentrate" on synthesizing just facial features.

## System Scripts

* **Brute force attacking script:** which runs the attack explained before. The script runs a brute force attack on the database it is given and outputs statistics regarding the attack.
* **Threshold determination script**: what the ideal threshold is, where we can say that 2 faces represent the same person. The DLib library recommends using a threshold of about 0.6, but since the database we use is considered "easy" (in the sense that all pictures were taken in the same format). we went ahead and tried to determine the threshold ourselves. The script, given a set of faces, plots the distances between two pictures of the same person (Shown in blue as freq1), and two pictures of different people (Shown in orange as freq2):

[](https://github.com/Royz2123/Biometric-Attack/blob/master/figures/threshhold500.png)

As can be seen here, the ideal threshold to minimize false-positives and true negatives lies around the 0.4 mark, since it sits in the middle of the 2 distributions.

## System Features

* The model database and the attacked database can be loaded from both a folder containing subject's faces, or a CSV file with the already extracted features
* A time-based recovery mode, so that a run can be retraced using a previous seed
* Visualization of the synthesized faces by averaging faces of people that were at a small distance from the sampled feature vector:

A screenshot of a person

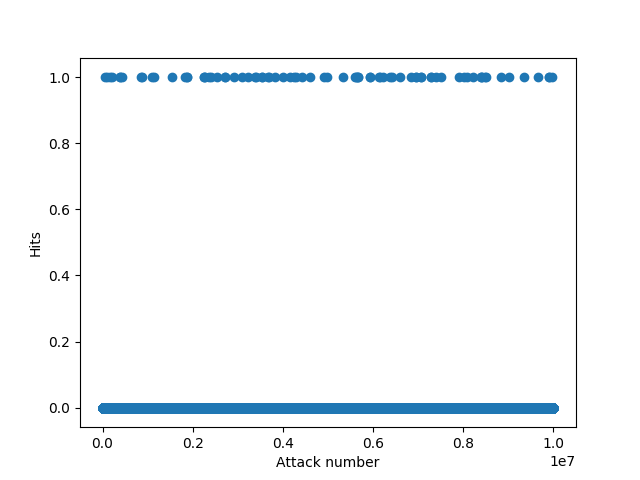
Description generated with very high confidenceA screenshot of a social media post

Description generated with very high confidence

# Results

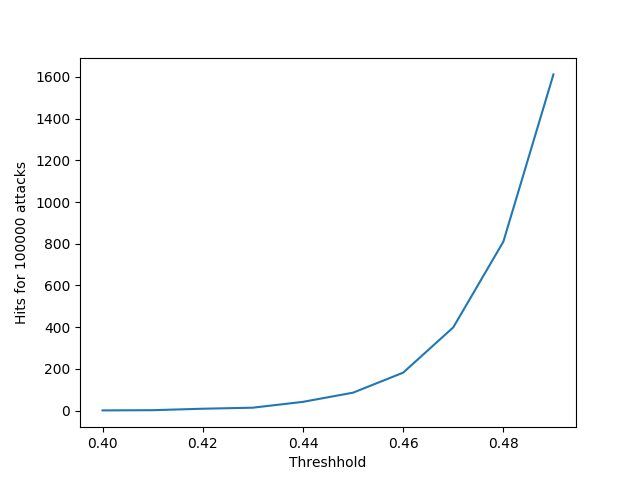
Once we were able to sample faces from the model, we were able to run several attacks on the testing database.

We found that for around 10 million synthesized faces, we had 80 hits (Around 0.001%, or 1 in 100,000 hit ratio for 0.4 threshold):

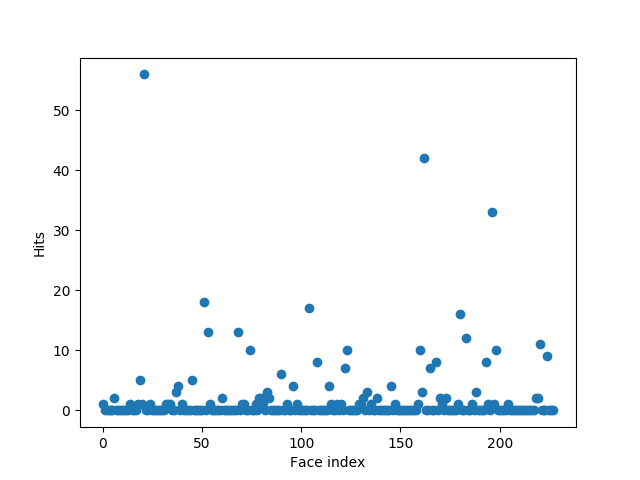
[](https://github.com/Royz2123/Biometric-Attack/blob/master/figures/diff_bases_10_mil.png)

This hit ratio was consistent also for 50 million synthesized faces, where we had 417 total hits (Details for this run can be found at general log file, from **05:06:2018\_18:41:57**). This was the largest attack that we ran.

Another interesting phenomenon is the rise in hits as a factor of the threshold. Turns out that if we increase the threshold by just a little, we suddenly reach many more hits:

[](https://github.com/Royz2123/Biometric-Attack/blob/master/figures/thresh_test4.png)

Another interesting figure that we created, was what faces were hit the most. Does an "average face" exist, that keeps getting matched all the time? We plotted the number of hits for an attack size of 50 million synthesized faces:



Here we can see that one face got over 50 hits, substantially more than the rest.

# Conclusions

In conclusion, we found that even with a harsh threshold (stricter than the one specified in the library), we still managed to synthesize faces that were close enough to ones in the database.

We conclude that if one was to use our technique as a brute force attack, his success rate would be **roughly** the same as brute-forcing a 6-7 digit code, which can be easily reached in seconds-minutes by any modern computer.

Our work helps get a better feel for the entropy of face spaces, and how faces in them are distributed.

We also found that if we would have used DLib's recommended 0.6 threshold, we would have gotten a much higher hit ratio. From this we conclude that tuning the threshold is crucial and is dependent on the database.

# Thoughts for the future

After showing that the security of this representation isn't safe, one way we could continue our project is to develop a more detailed representation for faces.

Another thought we had for the future was to try and reverse the feature extraction method, so that our program could synthesize actual pictures and not just vectors.

# Final Remarks

Throughout the project we learnt on how to use the DLib library, which was a completely new environment for us. We also got some hands-on experience working on the cloud, as we ran some of our attacks there.

On the theoretical side, we had some previous knowledge regarding PCA and the inverse sampling method, which we brushed up on. This project gave us an opportunity to implement topics we only learnt about theoretically in previous courses.

The main bottleneck for this project was getting the attack to work, since the code was very difficult to debug and test. Overall, we found the project to be both interesting and challenging.