University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Tomasz Garbus **Dominik Klemba**

Jan Ludziejewski **Łukasz Raszkiewicz**

Novelty face authentication with liveness detection using depth and IR camera

Bachelor's thesis
in COMPUTER SCIENCE

Supervisor:
mgr Grzegorz Grudziński

June 2018

Abstract

Currently, commonly used authentication methods are becoming obsolete, due to the lack of convenience, adversarial technology improvements or easy to make user errors. We believe that biometric authentication has the potential to become one of the safest, most convenient and most efficient methods. In this paper we describe our efforts to improve face recognition using a camera with depth-perception and infrared capabilities, as well as our search for new liveness detection methods such as skin detection using multispectral imaging. This text is mostly written with mobile solutions in mind, but the results could also be applied in other settings.

Keywords

face recognition, liveness detection, skin detection, depth camera, biometric authentication, machine learning, neural network, authentication

Thesis domain (Socrates-Erasmus subject area codes)

[11.3] Informatics, Computer Science

Subject classification

Security and privacy
- Security services
  - Authentication
    - Biometrics

Tytuł pracy w języku polskim

Nowatorskie uwierzytelnianie twarzą z wykrywaniem życia przy użyciu kamery głębi oraz podczerwieni

Acknowledgment

We would like to thank Samsung R&D Institute Poland for sponsoring this work. The views expressed in this paper are those of the authors and do not reflect the official policy or position of Samsung R&D Institute Poland.

Furthermore, we would also like to thank:

- our dissertation advisor, mgr Grzegorz Grudziński of Faculty of Mathematics, Informatics, and Mechanics at University of Warsaw, for supervising our work;
- Antoni Jakubiak of Samsung R&D Institute Poland for overseeing this project;
- Karol Brunejko of University of Warsaw's Legal Clinic for legal consultation;
- everyone who has contributed to the face dataset;
- and everyone who has helped with the creation of this thesis in any other way.

Contents

Chapter 1

Introduction

In the world of authentication, there is always a trade-off between convenience and security. We would like to compare commonly used authentication methods with regard to both factors and argue that face recognition is more than a good balance between the two – it does very well in terms of both security and convenience.

1.1. Conventional authorization methods

Passwords

Passwords come in various forms and cannot be easily placed on a security-convenience curve. Their strength is exponentially dependant on their length.

The strength and form of a password often depend on the context of its use. For instance, passwords used for unlocking the screen in a mobile device tend to be simple and quick to input – a pattern to draw, a 4 to 8-digit pin, in rare cases – an alphanumeric sequence. Passwords used for bank accounts, on the other hand, can be undeniably strong, to the point of being a randomly generated sequence of lower and upper letters, digits and special characters.

With those two extreme examples in mind, we can conclude that passwords can be very secure and they can be very convenient. But that does not imply that passwords can be very secure and very convenient at the same time.

Another thing to keep in mind is that passwords are an authentication method that relies on the user's *knowledge* – knowledge of the secret code and the ability to remember it at all times.

That, however, is changing. Password managers and credential managers have been taking the responsibility of not only storing the passwords but generating them too. The user does not have to know their password – it is sufficient that the password manager knows it and that the user is in *control* of password manager. It is a good defense against phishing attacks, assuming the user can be tricked into giving their password more easily than the password manager.

Hardware tokens and magnetic cards

Sometimes, to protect valuable resources, something stronger than a standard password is needed. Hardware tokens such as U2F-like tokens or password-protected RSA SecurID and magnetic cards are good choices then. Those authorization methods rely on the user's *possession* of said token or magnetic card. To gain access to the victim's resources, an attacker must gain possession of the physical key, whether it is a U2F- or SecurID-like token. Magnetic

cards are more vulnerable to attacks – it is possible to conduct an attack without obtaining the card itself, but instead being within its close range.

1.2. Biometric authentication

Conventional authorization methods are based on *possession*, *knowledge* or *control* over some resource. Usually, the resources in question are copyable. We have no intention of undermining such methods, because they can be extremely secure if designed well.

Instead, we would like to focus on another paradigm of authentication, which relies on *being* – being the right person to authorize. With this approach, the authorization keys – users – are unique and non-copyable. However, their uniqueness is as good as the authorization system's ability to distinguish between them.

Biometric authentication is a set of techniques striving to recognize a human being by their appearance, body or behavior.

Fingerprints

Fingerprints have been used for a long time for identifying their owner, especially in criminology. They are perfectly fine in terms of distinguishing people as everyone's fingerprint is unique. Moreover, even if there happened to be two people with identical fingerprints, it would be an extremely unlikely coincidence for one of them to try to hack the other. Unfortunately, while users are a non-copyable resource, their fingerprints are indeed copyable.

Behavioral biometrics

Behavioral biometrics have been making their way into the banking industry. They are a set of techniques supporting the conventional authentication, by learning, and verifying, user's preferences and habits, such as (by International Biometrics+Identity Association):

- Behavioral qualities of the input data, such as typing speed or touchscreen interactions and their timing;
- Supporting contextual factors of the current transaction, such as user device type, IP address, geolocation; and
- The user's historical behavioral factors, such as the typical timing of user access, prior purchasing or access patterns, etc.

The downside is, behavioral biometrics are not a standalone solution. They are more of an anomaly detection system reinforcing the authentication system.

Face recognition

Face recognition is an authentication method that is both convenient and secure – in theory. It has been used in smartphones, tablets and laptops for several years now, mostly to unlock the device, but never really trusted enough for securing more vulnerable resources. Faces, like fingerprints, are unique, but their uniqueness cannot be taken for granted. For instance, a static, two-dimensional picture of a face is perfectly copyable.

1.3. Liveness detection

To design a secure face recognition based authentication system, one should ensure the user's face is non-copyable. Only then will the system fully comply to our paradigm of relying on *being* instead of *possession*.

This can be achieved by adding liveness detection to the authentication process. Liveness detection itself can be based on multiple techniques.

Existing methods

Various mobile phone manufacturers have implemented methods attempting to detect whether the visible face is actually a human and not a photo. For example, users were asked to blink or smile to unlock their phones.

Such methods definitely increase the security of face recognition, but they are inconvenient to the user – ideally, one would not need to take any action to unlock their device – and were often easily hacked by having additional photos of the owner, even fake ones with the eyes being covered.

Depth channel

Using a depth channel in addition to a flat image is a form of liveness detection itself. When implemented well, such system should block all attempts to authenticate using a photo of an authorized person. However, it could still be broken using a detailed mask of that face. While that would require a lot of effort, it is a realistic way to break into for example a stolen phone of an important person.

Pulse detection

Determining a person's pulse from a video is not a new idea [2]. We have found existing implementations of programs that detected the user's pulse, using for example a webcam [3]. Unfortunately, we found out that such liveness detection method would be too slow for common authentication purposes. Detecting the pulse takes several seconds and requires the face to remain in one particular spot through the entire procedure. Therefore we we abandoned further research on this subject.

Skin recognition

The liveness detection idea which we decided to research is skin recognition. With a reliable method to detect whether real skin is visible in the camera, it should be nearly impossible to authenticate without actually showing the owner's face to the camera. Our proposed skin recognition ideas will be described later in this paper.

1.4. Results

In our work, we have focused on skin recognition and face authentication with depth channel and IR, because it does not depend on the surrounding environment (and has an additional upside of having the possibility to improve the photo quality [4]).

Skin recognition

We have developed several skin recognition techniques working with different sets of light wavelengths. We have also constructed a simple hardware prototype and experimented with machine learning approaches to per-pixel skin recognition.

Face authentication

We have constructed a depth + infrared face dataset and evaluated different machine learning approaches to face recognition. We have measured the effect of authentication time limit on its effectiveness and the trade-off between security and convenience.

Chapter 2

Skin recognition

Earlier, we described why liveness detection is an important part of a face recognition system and why it could greatly improve security. In our research, we focused on using light reflectiveness to recognize skin on photos. This idea comes from the fact that every material has its own, unique spectrum. Different types of human skin reflect visible light very differently [5], but those differences become smaller when going further into infrared wavelengths [6] [7], which is why we decided to use infrared light in our experiments. Ideally, we would like to have a spectrometer in a mobile device, which has been done before [8].

Additionally, with the information of which pixel contains human skin, it would be possible to use this to remove the background from face photos [9]. As described in a later chapter, this is an important part of face recognition, and is currently done using methods based on machine learning, which are usually rather slow. Using skin detection could significantly improve the performance of background removal.

2.1. Technical aspect – how does a camera work?

To think about the security aspect of skin detection, it is critical to understand how image sensors works. The most common sensors are *CCD* (*charge-coupled device*) and *CMOS* (*complementary metal-oxide-semiconductor*) [10]. Due to better quality, as well as sensitiveness, for security reasons it probably would be better to use CCD. However, both matrices have same problem – limitations on wavelength sensitivity. Such sensors are sensitive to wavelengths approximately up to 1050nm [11], which may be a serious limitation as human skin has more undifferentiated characteristics in greater wavelengths. Therefore looking for the the best possible skin detection method, it may be convenient to assume that other type of light sensitive matrices are in reach of an inventor. Nevertheless, we were limited to very common cameras.

We are describing one possible realization, but keep in mind that there are more solutions, which we did not find useful in our problem.

Monochrome

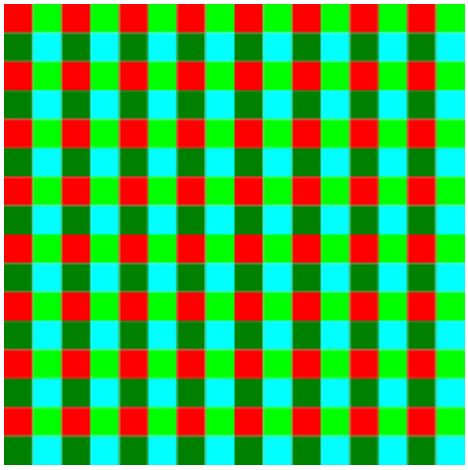
As an image sensor is a matrix built from many small photodiodes, it is natural to create a monochromatic image. Because photodiodes do not distinguish different wavelengths, only light intensity, it is hard to measure light reflectiveness. The only possible way to do that is to take a separate picture for every wavelength. It is not easy, as you have to provide your own lighting and somehow remove light coming from other sources. Also, all pictures have to

be made in a very short time interval, as the subject cannot move between shots.

Multispectral imaging

Multispectral imaging is a way to measure every wavelength intensity in one picture, so it is an answer to the problem of the object moving. Probably the easiest way to describe it is to look at an RGB camera (with a Bayer-like filter [12]). There, every photodiode is responsible for exactly one color, where as color we understand a wavelength interval (it is important to understand that we can consider any wavelength interval as one color). The result picture is built from small squares (mostly 2×2) of photodiode matrix data. In fact, there may be more than just three colors. For example, you may find a camera which can see two types of green. Usually, in the end everything is converted to the well-known RGB format.

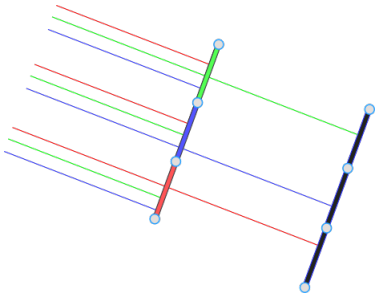
Figure 2.1: RGB image sensor matrix with two types of green – sensors arrangement.



Each sensor reports what is the light intensity to the camera, which then reports that intensity and which color that sensor filters to the software.

One of the possible realizations of reducing the range of light visible through a sensor is to take a standard image sensor, which may see the full spectrum of light (CCD – 350-1050nm [11]), and place filters in the way, so that only specific light will reach the chosen photodiode.

Figure 2.2: Visualization of a filtering mechanism.



Hyperspectral imaging

Hyperspectral imaging is creating pictures using a camera with the capability to distinguish a large amount of colors (even hundreds or thousands) instead of just a few. It would be

very helpful to have one in a skin detection device, but it would be impractical to put a hyperspectral camera in a mobile phone.

2.2. Technical aspect – how can it be created?

As we now know how a camera works, we can distinguish two ways of building a skin-detecting mobile device.

Monochrome

We can work without filters (or with one for the whole camera), but then we have to take a picture for every wave length. Also, we need one diode producing the exact wavelength we want. It is an easier method to implement in a phone, but taking pictures would have to be very fast.

The biggest advantage of that solution is the possibility of lighting with different LEDs in random moments of time, so hackers would not be able to play with their own diodes to modify the results.

Multispectral imaging

There is also a possibility to use filters with specific infrared colors. But it is important here to avoid standard smoothing algorithms used in cameras. The real light intensity from every sensor is needed, without any postprocessing.

2.3. Our attempts

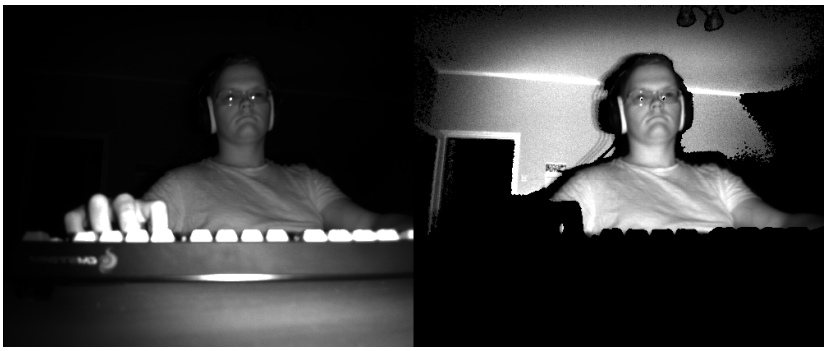
2.3.1. Detecting reflectiveness using Kinect

Our first idea was to use a Kinect v2 depth and infrared camera to detect how well does a surface reflect light. Kinect v2 cameras have their own source of infrared light and they make a very good job at filtering other sources of light. So, for each pixel on the infrared image, we know how much light coming from the Kinect's light emitter was reflected in that particular place. Also, Kinects are depth cameras, which additionally gives us the information on how far away from the camera is the object visible on that pixel.

Since the source of light is in the same device as the camera, the distance seen on the depth image is also the distance from the light emitter. This is an important observation, because we know how distance affects how much light arrives at a particular place. If we have a source of light that, at a given distance, lights up a 1cm $\times$ 1cm square of a flat surface with a particular amount of light, then at double that distance it will light up a 2cm $\times$ 2cm square with the same amount of light, which means that the same amount of light is distributed over a 4 times larger surface, so the intensity of light has to be 4 times lower than before.

With those observations, we took pairs of IR and depth images, and for each pixel calculated a new value of $ir\_intensity \cdot depth \cdot depth$, which should estimate how well light was reflected at that point, regardless of distance from the camera.

Figure 2.3: An unprocessed IR photo, and an image calculated using the method described above.



As seen on figure 2.3, we have accidentally created a night vision camera (please note that some of the black areas are there because Kinect cameras do not give depth data for objects closer than 50cm) – but this means that our idea is to some extent working, because the point of it was to make objects in the distance indistinguishable from those close to the camera.

However, one thing that is not considered in that method is that objects can also be at different angles to the camera. A sheet of paper perpendicular to the source of light will reflect more light to the Kinect than it would reflect if it was at a 45 degrees angle.

Figure 2.4: Three IR images processed with the described method, with a manually determined interval of values marked red.

We have manually selected an interval of values and marked them red, which can be seen on figure 2.4. On the first two pictures there, the hand is at the same angle, but at a different distance to the camera. It is visible that the calculated values remained very close regardless of the distance, which was a success. However, on the third picture, the hand is at a different angle and that changes how it reflects light towards the camera, which makes the values different.

Knowing the depth value and certain properties of the camera, it is possible to calculate the 3D coordinates of each pixel. We calculated cosine between vectors using the formula $\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{||v_1||\ ||v_2||}$. We tried to calculate reflectiveness using two data points: how flat is the surface (using neighboring pixels and calculating angles between them), and what is its angle towards the camera (using a basis vector).

With that information, it is possible to calculate the angle towards the camera between each two pixels and the camera, and then use that value in the method above to make it independent of angles. However, our attempts to do this were unsuccessful, possibly because the depth camera is not precise enough.

2.3.2. Using three infrared wavelengths

Prototype

With the previous method using only one light wavelength (the one emitted by Kinect), we decided to take photos using three different wavelengths. This gives the opportunity to analyze how does the way the object reflects light changes with regard to light wavelength, instead of directly looking at how bright a point is.

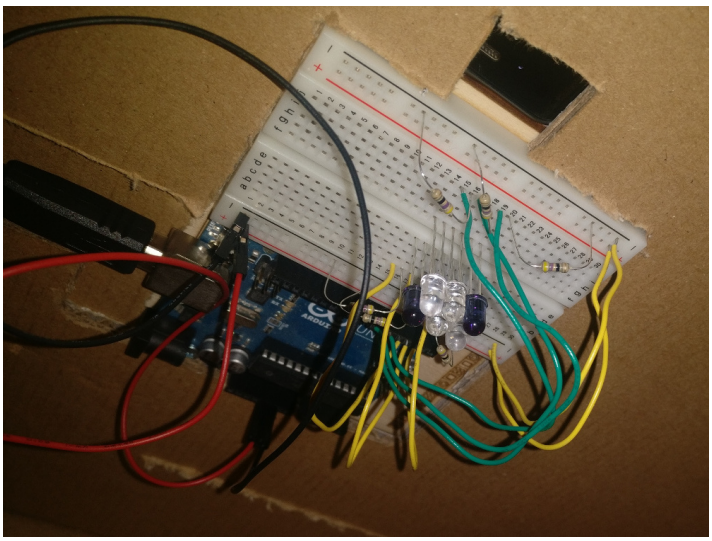
To make any research, we had to take three photos of the same object reflecting three different wavelengths of infrared light. The camera, the infrared light sources, and the photographed object had to be in the same position for all of the photos. An ideal way to do this would be to use a multispectral camera, but they are very expensive and we were not able to use one.

One possible way to take a photo of how a certain wavelength is reflected would be to use a filter that only lets that wavelength to pass through it. However, changing filters on the camera between taking photos would take a lot of time, which would create a risk of changing the relative position of the object and the camera, which is unacceptable.

We borrowed a mobile phone which had a filter mounted on its front camera that allowed various wavelengths of infrared light, but not visible light. This opened up the opportunity to take a different approach – instead of filtering the selected wavelength, we want to illuminate the object using only that wavelength.

We purchased diodes emitting light in the following wavelengths: 850nm, 890nm, 940nm. Those diodes were connected to an Arduino board. We used two diodes of each wavelength, and mixed their positions to make sure that the centers of sources of light at each wavelength are as close as possible.

Figure 2.5: Arduino board with six IR diodes.



Since taking all three photos at the same time would be impossible, we stabilized the camera and the diodes by putting them in holes in a cardboard box.

Figure 2.6: Outside view of the Arduino board and the camera.

With this setup, all that remained to do was to take photos. The camera and the phone were stable, but when photographing for example a human hand, it might move. The photos have to be taken in the smallest possible intervals and human interaction can not be required while taking them, because that could result in moving the camera or the diodes.

Turning specific diodes on and off was a rather easy task. Arduino is by definition programmable, so we just wrote a program that listened to data sent on a USB cable and turned on the requested diode.

We found an Android app [13] designed to take photos remotely when commanded so from another phone with a special pilot app. It is open source, so we read its source and found out that it just sends simple LAN broadcast messages.

With this information, we were able to write a Python script that did the following:

- connect to Arduino through USB, turn on 850nm diodes
- wait 0.75 seconds
- send a broadcast message to take a photo
- wait 0.75 seconds
- turn on 890nm diodes
- wait 0.75 seconds
- take a photo
- ... – analogical procedure to take a photo with 940nm light

Since we were using only the front camera from a relatively old phone, we had to take a 1.5 seconds break between each photo – otherwise there was a significant chance of the camera app not catching one of the broadcasted commands and taking only two photos. The time between the first and last of three photos was 3 seconds, and that was an acceptable delay to make sure that the photographed object was in a stable position.

Figure 2.7: Photos taken with 850nm, 890nm, and 940nm light respectively.

Detecting skin – preprocessing

As we only had a phone camera (with a not-fully-known filter), our input was a picture-matrix in which every pixel was a nine elements tuple. In fact, values there were not independent, because for almost every pixel in all JPEGs, $R \approx B \approx 2 \cdot G$ occured. Therefore, using all that data would only cause overfitting, instead of actually improving the model.

We had many preprocessing ideas, all of which were focused on using every pixel independently. The one with the best results was taking a few values from each of three pictures and calculating differences concatenated with the basic vector (best results for random forest, SVM worked best with vectors normalized to unary vectors). There is a chance to improve the preprocessing by using more than one pixel, but we were focused on getting more reliable results (taking an average of adjacent pixels would make it easier by reducing the amount of anomalies).

Detecting skin – models

Unfortunately, handmade algorithms to detect skin failed, so we decided to use machine learning. We prepared an image mask for a picture of a hand, which provided information to the algorithm about which pixels are skin and which are not.

Figure 2.8: Example of a manually made skin mask.

We used `scikit-learn` to test different models. We hoped that SVM would work well, and with a linear kernel it did show some promise.

Figure 2.9: SVM without keeping only the well-lit area of the picture.

However, Random Forest achieved better results, as seen on the images below.

Figure 2.10: Skin detection with SVM using preprocessed data. The first two pictures come from the learning set, others are from the test set.

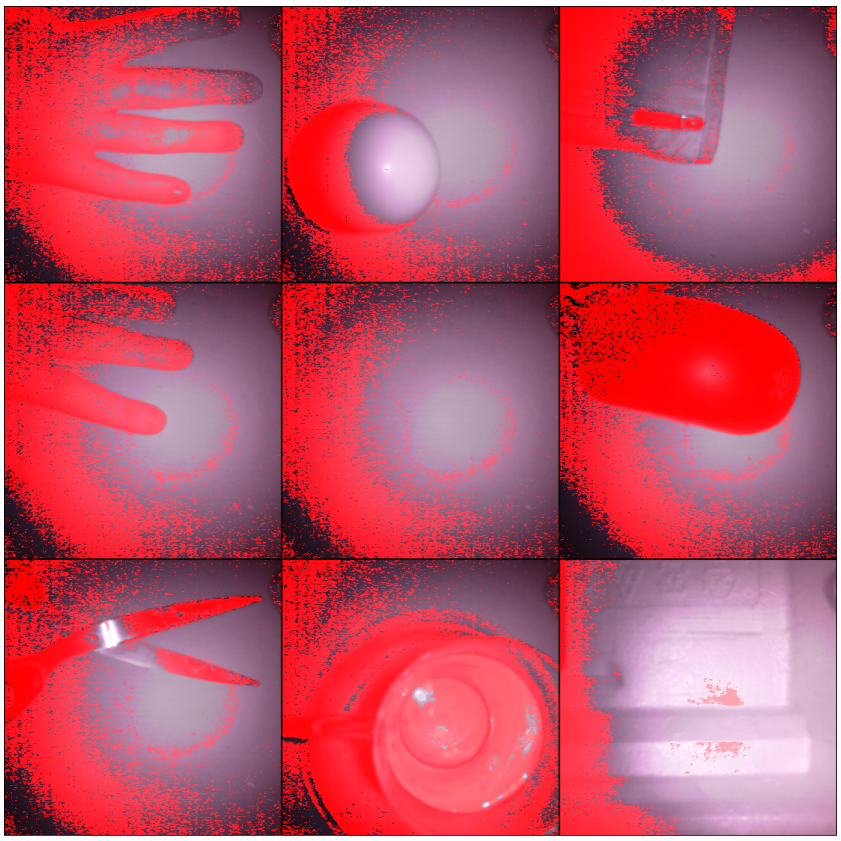
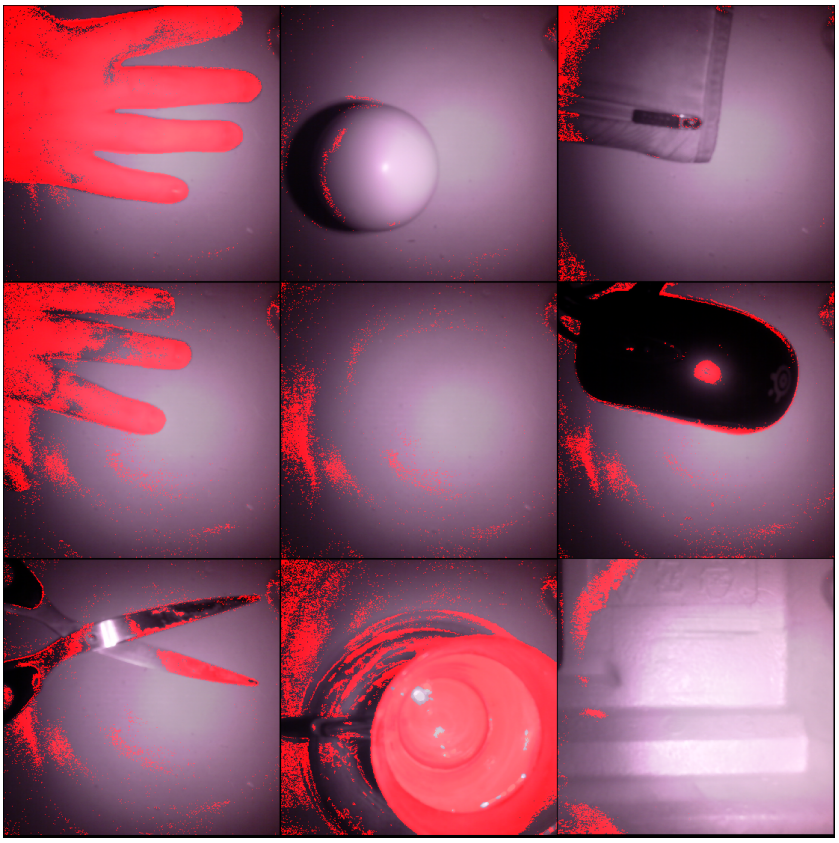
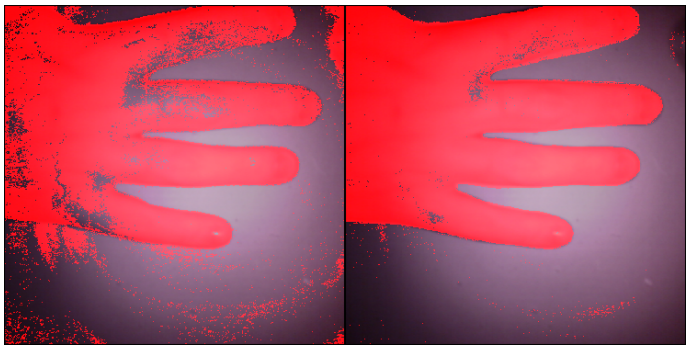


Figure 2.11: Skin detection with random forest using preprocessed data. The first two pictures come from the learning set, others are from the test set.



It is important to notice that a lot of information was lost in the picture that we had, as we were not able to take RAW photos or at least avoid JPEG compression. The consequences of that were visible very well when we tried to intentionally overfit out model on some images, and we failed because there was not enough information even to overfit. Therefore we consider our results better than expected.

Figure 2.12: Overfitting random forest on one picture. Wihout and with preprocessing.



2.4. Results

The results achieved by our skin detection were far from perfect, however we still think that using infrared photos to analyze surface reflectiveness and using that to detect skin is a promising method. We were very limited by the low quality and JPEG compressed photos, overall lack of high quality hardware, and our lack of knowledge in electronics, but were still able to achieve results that visually make sense. With more resources, we believe this could increase security of face authentication solutions.

The next big step associated with development of that skin detection technology may be using a hyperspectral camera to find a few (for example three or four) colors giving greater security, as well as measuring quality of that solution with a fixed number of colors. Unfortunately, we were not able to use a high quality hyperspectral camera to measure possible effectiveness and security of such solution.

2.5. RGB skin detection simulation

Because we wanted to have the ability to check how skin detection (if it was implemented in a possible future device) could be utilized by other algorithms, but we could not integrate our prototype with anything else, we prepared a skin detection heuristic based on RGB images. We based our algorithm on a transformation from *Human Skin Detection by Visible and Near-Infrared Imaging* [7]:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

For (Y, Cb, Cr) values calculated like that, we marked pixels as containing skin when $Cb \in [77; 127] \wedge Cr \in [133; 173]$ held [7]. However, those conditions gave us too many false positives (what is understandable, as it was a universal solution), so we decided to improve the decision process.

For each pixel in an image, we calculated its mark, where a mark of a pixel $p = (R, G, B)$ is the vector (Y, Cb, Cr). After that we took a few marks of pixels from the central part of image and sorted them by euclidean norm. Heuristically, we assume that the middle element of the new table is a skin pixel. At the end, we consider a pixel as a skin pixel if and only if its mark was close enough to the middle mark from the sorted list.

Figure 2.13: RGB skin detection heuristic.

Chapter 3

Face authentication

Face recognition is a problem that has been around for a while and human-level solutions have been developed, such as FaceNet [14]. However, there are many subtleties to face recognition in mobile devices security, in particular when using unconventional cameras.

From the security context, false positives are very problematic. This observation alone poses two questions:

- How to measure a model's performance to punish false positives?
- How to maximize our chosen score measure?

We have tackled two problems in our research – multi-class and binary classification. Binary classification is more adequate to the possible applications of our research. Multi-class classification allows the reader to compare the achieved results with other face recognition models.

The second observation is more optimistic: the front camera of a mobile is capable of capturing relatively many frames per second. We decided not to constrain our methods to a classification based on a single frame. Moreover, a large training set can be built when adding a new user to the device – it is not unreasonable to make that procedure longer for increased security. We aimed to take advantage of those factors.

3.1. Data set

We have set requirements for our dataset, aiming to reflect real-life appliances:

- **depth and IR channels**: in real-life appliances, leveraging the infrared camera allows equally good vision regardless of lighting conditions.
- **various vertical angles**: when a user is looking at their mobile device, their head may be rotated vertically to a various degree. However, it is reasonable to assume that the horizontal rotations will be small, especially if the user is consciously trying to unlock the device.
- **many frames per subject**: when a new user is being added to a mobile device's authentication system, it is acceptable to require them to look at the camera, at various angles, for a short period of time. Many frames can (and should) be taken.

There are several good databases for face recognition with depth camera: The EURECOM Kinect Face Database [15], RGB-D Face database [16], The Florence Superface dataset [17].

However, none of them was deemed suiting for our project, primarily due to the lack of the IR channel. Thus, we have decided to build a dedicated dataset.

Data collection

Hardware, libkinect

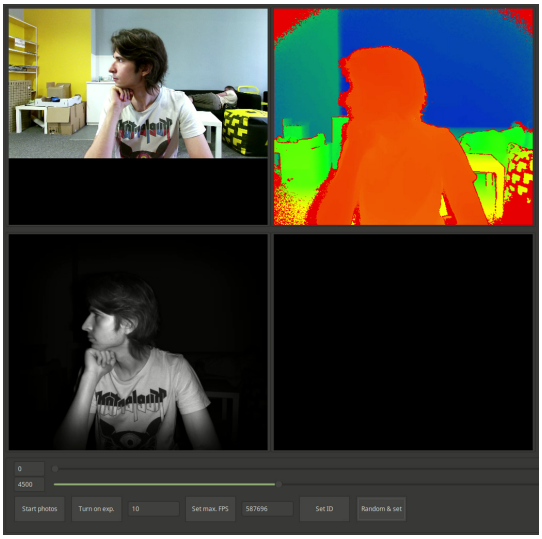
For this project, we had one Kinect v1 camera and two Kinect v2 cameras. Because we use Linux, we could not use Microsoft's API for Kinects and decided to use open source libraries: `libfreenect` [18] and `libfreenect2` [19] developed by the OpenKinect community.

Since our team had four members, we could not always have the same Kinects available for everyone, and sometimes one person would need to use Kinect v1 and another one Kinect v2 to run the same program. That is why we decided to create `libkinect`, which is a library that gives a simple interface for receiving images from a Kinect camera regardless of its version, allowing the programmer to focus on important code instead of wasting time on making the Kinect work.

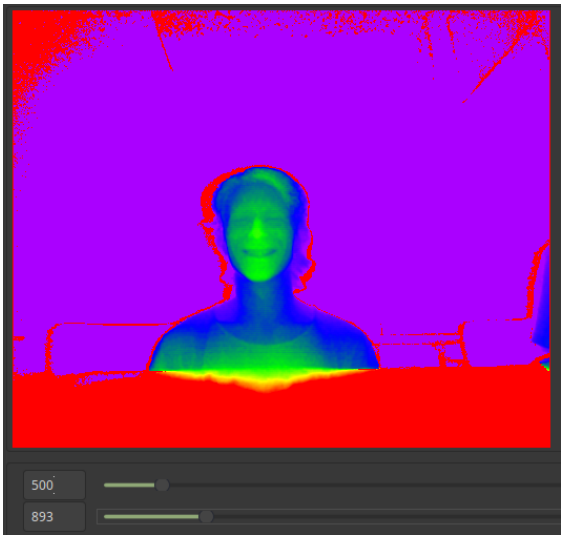
Based on `libkinect` and its file format for depth and IR photos, we made a program that displayed live video from the connected Kinect device and allowed saving received frames, a program that displayed a file, and a thumbnailer that can be configured to show thumbnails of depth and IR photos in graphical file managers.

Figure 3.1: Screenshots of implemented tools

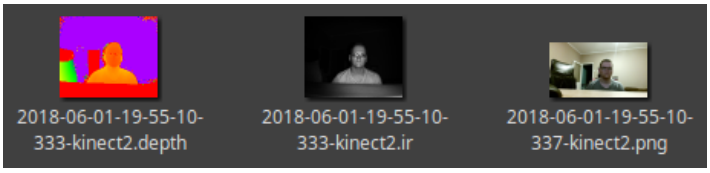
(a) Live Kinect display



(b) Depth file display



(c) Thumbnails in Nemo file manager



Both the live and file display programs allowed setting a scale for applying the color map to the depth image. On 3.1.a) the interval of shown depth values is very wide, so objects in the background are visible, but it is not possible to see any details on the face. On 3.1.b) face details are clearly visible because the interval is much more narrow.

Other features of the live display program included:

- Limiting frames per second while maintaining synchronization between depth and infrared frames – we did not want to collect photos at 30 FPS, because that would take too much disk space while not adding too much new information.
- Saving received frames to hard drive.
- Quickly changing the directory of saved photos to allow easy distinction between different volunteers.
- A fourth display, black on the screenshot, on which we could display experiments such as on figure 2.4.

Details of our dataset

The dataset consists of 44 subjects: 36 males and 8 females, mostly aged 20-25.

Each subject has been asked to follow an object with their entire head, making two cycles of slow, continuous movements: `up` → `center` → `down` → `center` → `left` → `center` → `right` → `center`. That procedure took around 20-30 seconds per subject. We believe it could be proposed as a part of a mobile device's first time security configuration.

We used only Kinect v2 cameras, because they give more detailed depth pictures than Kinect v1, and its infrared images do not have a dotted pattern, which is projected by Kinect v1 because it uses a different depth measurement method.

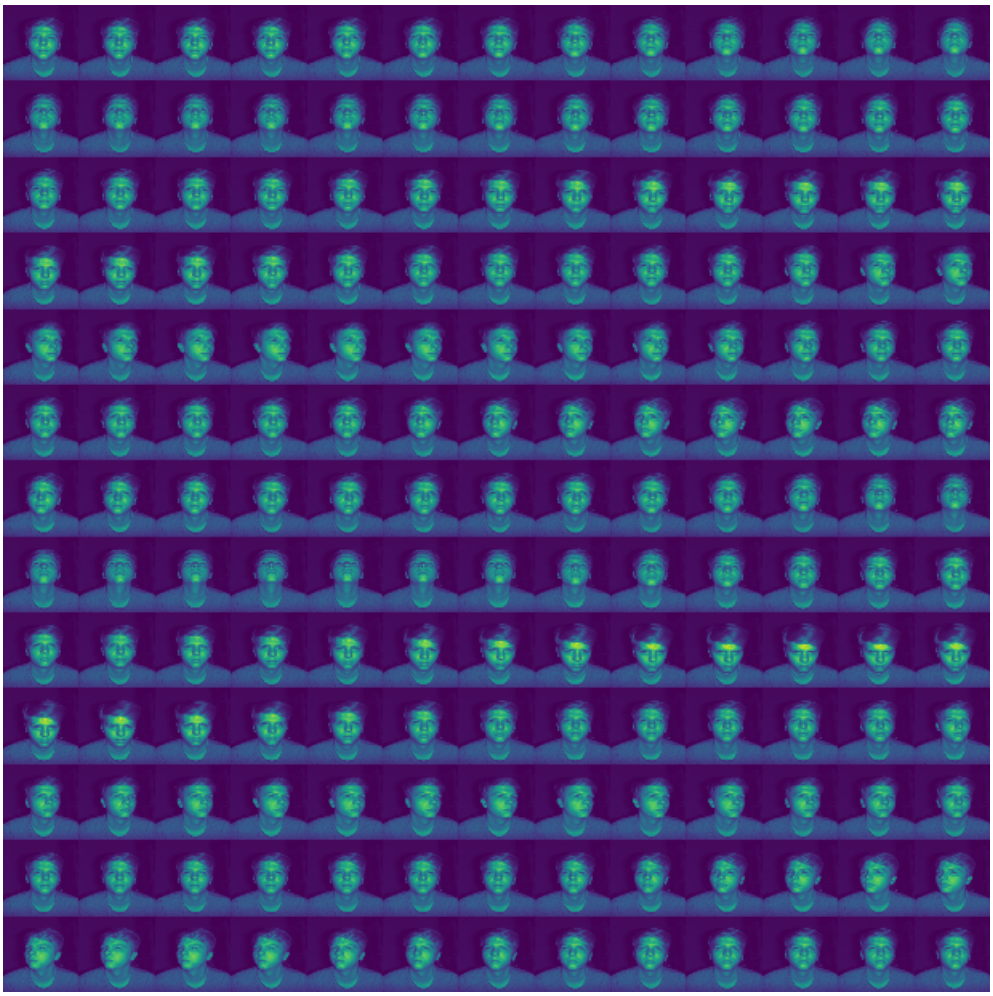
IR and depth frames are synchronized, taken at a 10 FPS rate. The dataset also contains RGB frames taken at a 10 FPS rate, not synchronized with IR and depth frames.

Test set

For the test set, we have chosen approximately first 25% of frames from each subject – the first two vertical head rotations.

Samples

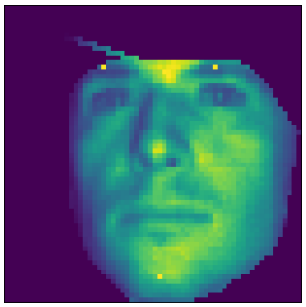
Figure 3.2: One of the subjects from the database – the first 269 IR frames.



3.2. Face angle detection

Using a depth camera, we can extract more structural features of an image, and normalize the face angle. The face is positioned in a cube, and the position vector is computed by finding three points located on the face, describing the surface and computing its orthogonal vector. We define face position as the angle between the computed vector and vector $[0, 0, 1]^T$ pointing towards the camera. We have chosen the points as the middle of each eyebrow and the middle of the chin and computed them with the same library we use for face trimming (as an average between chosen subsets of points). It is worth mentioning that choosing points too close to the edge of the face detected on a 2-dimensional image (as most algorithms do) might make the algorithm unstable, because a small error can place the point on the background, far away in third dimension.

Figure 3.3: Detecting the angle of a face. The face's orthogonal vector is positioned between eyebrows, face points are marked yellow.



3.3. Normalization

Depth mean and standard deviation normalization

Let F be the set of all points belonging to the trimmed face and $d(p)$ – depth of the point p. Let μ denote the mean depth, i.e. $\mu = \frac{\sum_{p \in F} d(p)}{|F|}$ and σ – standard deviation, i.e. $\sigma = \sqrt{\frac{1}{|F|} \sum_{p \in F} (d(p) - \mu)^2}$. For each pixel p, its new depth is calculated as:

$$d'(p) = \begin{cases} d(p) - \mu & \text{if } |d(p) - \mu| \leqslant 2 \cdot \sigma \\ 0 & \text{otherwise} \end{cases}$$

Lastly, all depth values are scaled to the interval [0..1]:

$$d''(p) = \frac{d'(p) - min(\{d'(r) \mid r \in F\})}{max(\{d'(r) \mid r \in F\}) - min(\{d'(r) \mid r \in F\})}$$

Face angle normalization

Depth channel provides substantial information about faces' geometry, thus allowing a meaningful normalization method – rotation to the frontal position.

With our ability to detect face angle (3.2), we can obtain three values: $\theta_x, \theta_y, \theta_z$ – angles along each axis.

The face can be represented as a cloud of 3-dimensional, "colorful" points. Point p is defined as a vector $[x_p, y_p, z_p, ir_p]^T$.

Let $R_x(\theta_x), R_y(\theta_y), R_z(\theta_z)$ denote the rotation matrices for X, Y, Z axes, as follows:

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix}$$

$$R_y(\theta_y) = \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix}$$

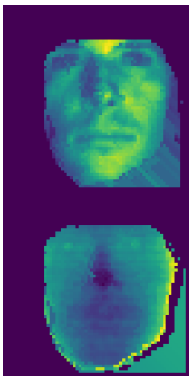
$$R_z(\theta_z) = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The operation of rotating a single point is a product of its coordinates and the three rotation matrices:

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ ir_p \end{bmatrix} \cdot \begin{bmatrix} R_x & 0_3^T \\ 0_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_y & 0_3^T \\ 0_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_z & 0_3^T \\ 0_3 & 1 \end{bmatrix}$$

Face angle normalization was implemented with smaller datasets in mind ([15], [16]). Our final training set contains many different angles of each face, rendering this technique obsolete. In our final models we have not used it.

Figure 3.4: Face from angle detection subsection, transformed.



3.4. Preprocessing techniques

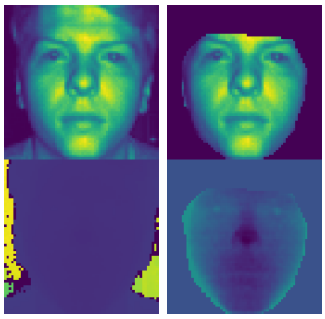
We have used several different preprocessing techniques.

Trimming faces

During the process of collecting the dataset, each subject has been recorded in only one set of clothes (the ones they were wearing at the time), one hairstyle, and the background may slightly vary among the subjects (the background behind subjects recorded at 12 a.m. is likely to be brighter than behind the subjects recorded at 4 p.m.; also, we were taking photos in 4 different places).

Since a classifier might leverage those factors, photos have been trimmed to polygons containing the faces. Such polygons have been found with Face Recognition library [20] on IR photos. Depth photos have been trimmed accordingly.

Figure 3.5: Before and after trimming the face.



HOGs

To transform 2-dimensional images into 1-dimensional feature descriptors, we used standard preprocessing called Histogram of Oriented Gradients, commonly used in object detection (Dalal and Triggs), purposely to ensemble 1-dimensional classifier with CNN, as different techniques achieve better results together. Briefly describing, HOGs divide image into cells, and then count gradient occurrences in each one. Recently, this technique alone has achieved sensible results in face detection using depth camera (Goswami et al.).

Entropy maps

To exctract meaningful information (as Goswami et al.), we used entropy maps, to amplify small variations on depth images, and provide more information to classifiers.

Channels vs concatenation

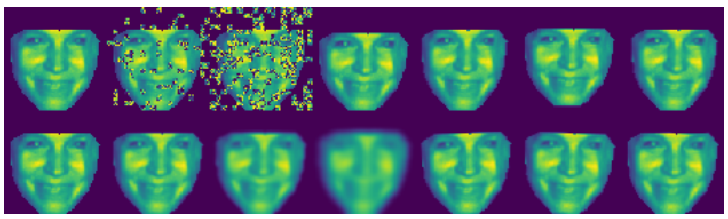
A choice of the input format passed to a classifier is not only a matter of images to include. Another decision to be made is whether different channels (IR, depth) and different preprocessed images (entropy maps, HOGs) should be concatenated or layered as channels. In our experiments, we have included both approaches, although channels quickly proved to give better results.

Data augmentation

To further enhance the training set, we have used image augmentations, with `imgaug` [23] library. The used transformations include:

- **coarse salt and pepper** – randomly placed patches of grain (black and white pixels)
- **padding** – shifting the input image 3 pixels in random direction
- **Gaussian blur** with several different strengths
- **rotations** by 2 degrees
- **PiecewiseAffine** – an augmenter that places a regular grid of points on an image and randomly moves the neighborhood of these point around via affine transformations.

Figure 3.6: Trimming + all possible augmentations.



3.5. Classifiers

3.5.1. Convolutional Neural Network

We have used a convolutional neural network with the following structure:

- **Layers**:
  - **conv. layer 0**: 20 filters, 5x5 kernel
  - **max pooling**: 2x2 pool size, 2x2 strides
  - **conv. layer 1**: 20 filters, 5x5 kernel
  - **max pooling**: 2x2 pool size, 2x2 strides
  - **conv. layer 2**: 40 filters, 5x5 kernel
  - **max pooling**: 2x2 pool size, 2x2 strides
  - **dense layer** with sigmoid activation function
- **Loss function**: Log Loss
- **Optimizer**: SGD
- **Kernel initialization**: Since we have experimented with deeper networks too, and each model was trained from scratch, a reckless initialization would hamper the models ability to learn and increase the number of iterations needed for convergence. To ensure a quick start of the learning process, we decided to initialize each kernel with **random normal distribution with** $\mu = 0,\ \sigma = \sqrt{(2/N)}$, as advised by He et al..

Figure 3.7: Convolved input images (only IR channel). First two rows are outputs from first and second convolutional layers. Third and fourth row are outputs from all 40 filters from the last convolutional layer.

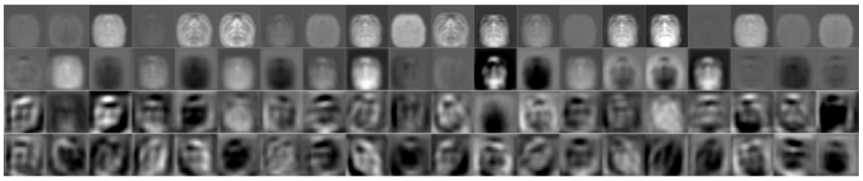
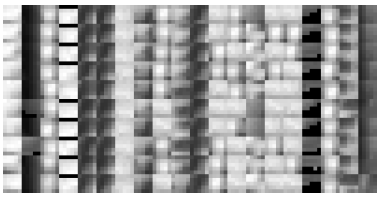
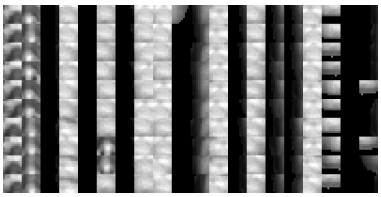


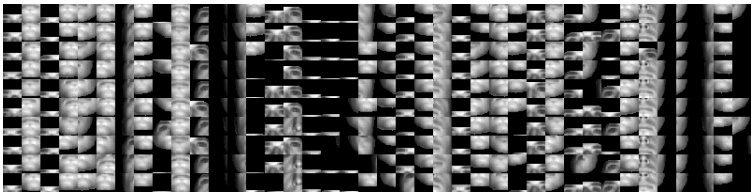
Figure 3.8: For each filter in each convolutional layer, 10 patches among one of the test batches were chosen that activate those filters the most. Horizontally, consecutive filters are presented, vertically – patches sorted from the highest activation. Note that for each patch, only the IR channel is presented.



(a) conv. layer 0



(b) conv. layer 1



(c) conv. layer 2

3.5.2. HOG + SVM

As an alternative to CNN, to classify HOG feature descriptors, we tested two common (Dalal and Triggs, Goswami et al.) approaches: Support Vector Machine, extremely randomized decision trees (Extra Trees), and both ensembled. Initially, while collecting the data and working with a smaller dataset, Extra Trees were achieving higher results, and SVM was highly overfitting, but as the work proceeded, the second approach was getting better and has finally overcome Extra Trees to the point that the other classifier was not even contributing to ensembling, so further work was continued only with Support Vector Machines. In multi-class classification, we used One vs Rest approach.

After tuning the parameters on the final dataset using grid-search approach, the following configuration of the most important parameters was chosen:

- **Polynomial kernel** with degree 3. Other kernels got significantly worse results.
- **Penalty C parameter**: 10, locally constant results results in range [1, 10].
- **Gamma parameter**: 0.7, locally constant results results in range [0.1, 1].

3.5.3. Ensembling

We have experimented with a simple ensembling method – voting. Each of our classifiers is implemented in a way that allows getting, apart from predictions, a vector of probabilities for each class.

Let $C^{(1)}, C^{(2)}$ be two classifiers running in the same mode – either binary or multi-class. Let $p_C(f)$ denote predicted probabilities for each class by classifier C, given input f.

An ensembling classifier is configured by two values w_1, w_2 – significance weights for classifiers $C^{(1)}, C^{(2)}$. Its output for input f is defined as:

$$C_E(f) = \frac{w_1 \cdot p_{C^{(1)}}(f) + w_2 \cdot p_{C^{(2)}}(f)}{w_1 + w_2}$$

We will use notation $C^{(1)}\ w_1\ :\ w_2\ C^{(2)}$ to denote ensembling of classifiers $C^{(1)}$ and $C^{(2)}$ with weights w_1, w_2f.

3.6. Quality measures

Models have been trained in two modes: multi-class and binary classification. Let T be the test set, p_t – predictions for test input $t \in T$ and y_t – ground truth for test input $t \in T$.

Accuracy

For classifiers trained in multi-class mode the accuracy score has been measured:

$$acc(p, y) = \frac{|\{t \in T \mid p_t = y_t\}|}{|T|}$$

Recall for fixed precision

When testing the model in binary mode two decisions are being made:

- **positive class (device owner)**: One subject from the database is chosen as the "device owner". All photos of this subject are then labeled as positive. All photos of other subjects are labeled as negative.

- **desired precision threshold:** Precision is defined as:

$$prec(p, y) = \frac{\{t \in T \mid y_t = p_t = 1\}}{\{t \in T \mid p_t = 1\}}$$

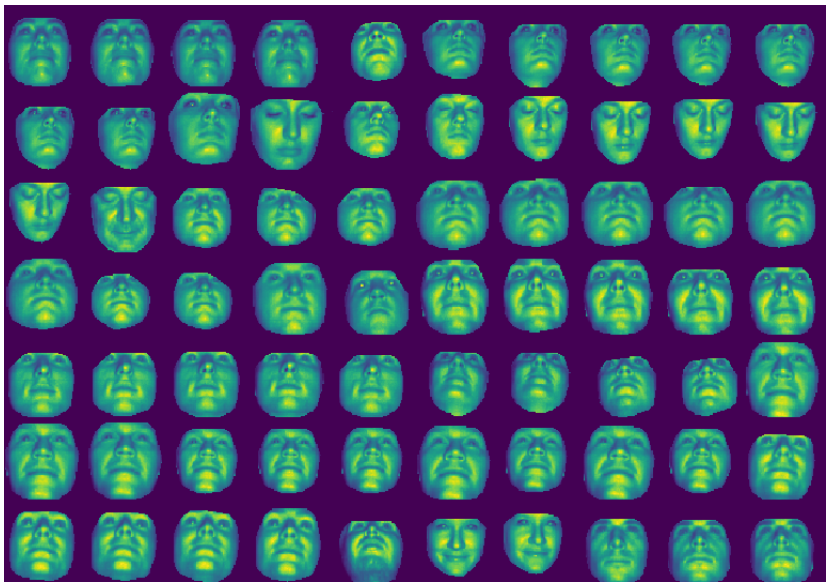
We have chosen 3 different precision thresholds: 0.99, 0.999, 0.9999.

Recall is defined as:

$$rec(p, y) = \frac{\{t \in T \mid p_t = y_t = 1\}}{\{t \in T \mid y_t = 1\}}$$

For each precision threshold, we have calculated the corresponding recall measure and averaged the result over 5 random choices for positive class.

Figure 3.9: Test samples misclassified by CNN



3.7. Single-frame model results

We call a model single-frame if it classifies the person on the photo using only one frame. We have evaluated several single-frame models to choose one multi-class and one binary classifier.

3.7.1. Multi-class

Table 3.1: Selected models performance in multi-class mode

Model	CNN inputs	Accuracy
CNN	channels	0.9721
CNN 10 : 1 SVM + HOGs	channels	0.9721
CNN 2 : 1 SVM + HOGs	channels	0.9648
CNN 10 : 7 SVM + HOGs	channels	0.9631
CNN 1 : 2 SVM + HOGs	channels	0.9536
SVM + HOGs	-	0.9099
CNN	concat	0.8899

3.7.2. Binary

Table 3.2: Selected models performance in binary mode

Model	CNN input	Precision	Recall
CNN 1 : 5 SVM + HOGs	channels	0.99	0.9079
		0.999	0.9013
		0.9999	0.9013
SVM + HOGs	-	0.99	0.8947
		0.999	0.8947
		0.9999	0.8947
CNN	channels	0.99	0.8487
		0.999	0.8487
		0.9999	0.8487
CNN 1 : 1 SVM + HOGs	channels	0.99	0.6908
		0.999	0.6777
		0.9999	0.6777

3.8. Multiple frames based classification

We have chosen CNN as multi-class classifier and CNN ensembled with SVM (CNN 1 : 5 SVM) as the best performing single-frame classifiers.

3.8.1. Generating predictions from multiple frames

Let n be the number of frames, C_1 – a single-frame classifier and $f_1, ..., f_n$ be the n consecutive frames. Let $p_{C_1}(f)$ denote the vector of probabilities predicted for each class by C_1, with frame f as input.

Note that $p_{C_1}(f) \in [0..1]^D$ where $D = 1$ for binary classification and D equals to the number of classes in multi-class mode – in our case $D = 44$.

The multi-frame classifier C_n can be defined as a function (namely arithmetic mean) from a tuple of n frames to the vector of predicted probabilities:

$$C_n(f_1, ..., f_n) = \frac{\sum_{i=1}^{n} p_{C_1}(f_i)}{n}$$

Another way of looking at multiple frames based classifier is as an ensembling of n identical classifiers, but each receives a different frame as an input.

3.8.2. Results – multi-class

Figure 3.10: Frames-accuracy relation for multiclass problem

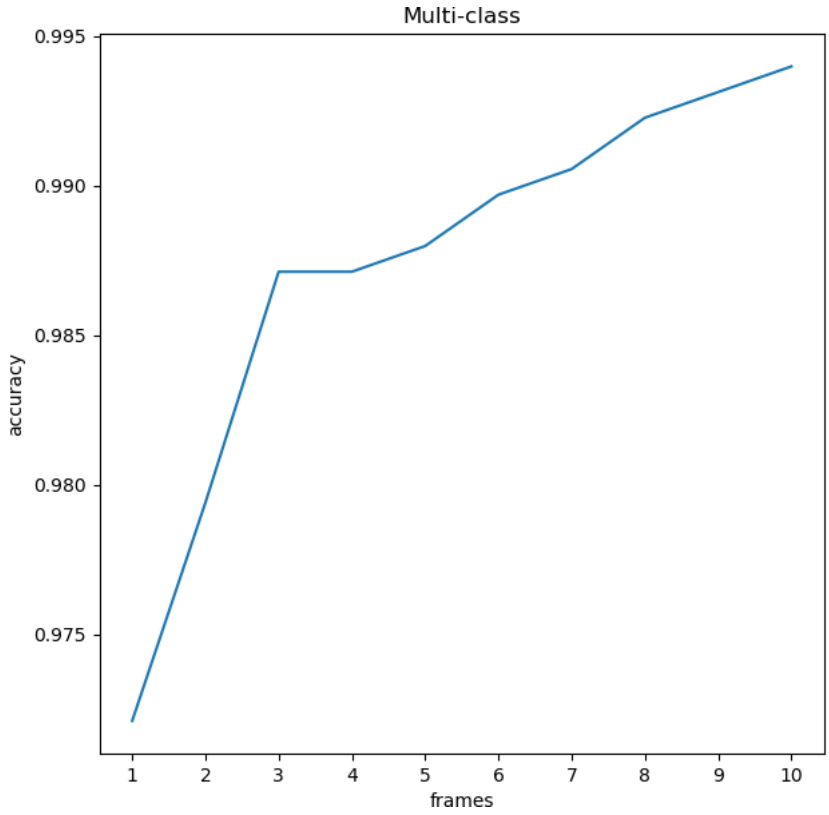


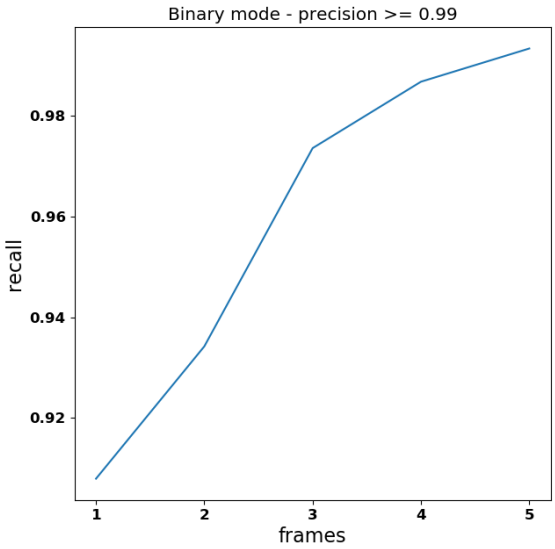
Table 3.3: Accuracy obtained by multi-frame based classification with CNN

Frames	Accuracy
1	0.9721
2	0.9794
3	0.9871
4	0.9871
5	0.9879
6	0.9896
7	0.9905
8	0.9922
9	0.9931
10	0.9940

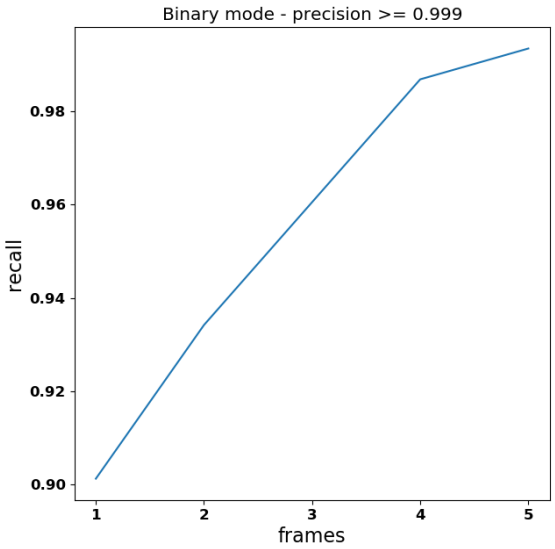
3.8.3. Results – binary

Figure 3.11: Frames-accuracy relation for binary problem

(a) Precision threshold $\geqslant 0.99$



(b) Precision threshold $\geqslant 0.999$



(c) Precision threshold $\geqslant 0.9999$

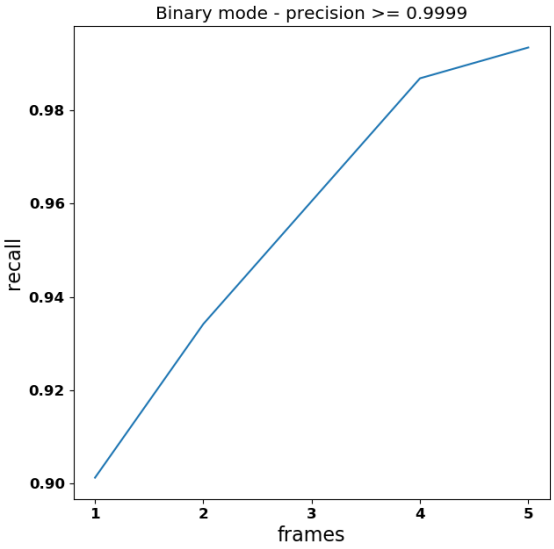


Table 3.4: Recall obtained by multi-frame based classification with CNN 1 : 5 SVM

Frames	Precision	Recall
1	0.99	0.9079
	0.999	0.9013
	0.9999	0.9013
2	0.99	0.9342
	0.999	0.9342
	0.9999	0.9342
3	0.99	0.9736
	0.999	0.9605
	0.9999	0.9605
4	0.99	0.9868
	0.999	0.9868
	0.9999	0.9868
5	0.99	0.9934
	0.999	0.9934
	0.9999	0.9934

Possible improvements

Not tested approach to multi-frame classification

Instead of implementing multiple frame based classification as an ensembling of several single-frame classifiers, a single classifier could also be used. We believe that a recurrent neural network (RNN) might be a good fit for the task.

Speed optimization

In our experiments, an external, heavy duty machine learning-based library has been used for finding and trimming faces (3.4). This has resulted in a single frame classification time of 260 milliseconds – most of which was spent on detecting and trimming the face. This is too slow for the 10 FPS rate.

We believe that a fully developed skin-recognition method, possibly with additional infrared channels on different wavelengths, could replace the all-in-one face detection library. Unfortunately, due to hardware limitations, we were unable to verify this assumption in our research.

3.9. Proposed solution

The solution we propose is a voting ensembling CNN 1 : 5 SVM, with classification based on 4 consecutive frames. We believe that 400 milliseconds is an acceptable authentication time. Using 4 frames, precision rate 0.9999 and recall rate 0.9868 are achievable.

In other words, our solution will fail to protect users resources or device only once in 10 000 cases. Out of 10 000 cases when the right owner tries to access the device or resources, only 132 times they will fail to authorize.

Chapter 4

Conclusion

Face recognition based authentication methods have been around for a while. However, they have always been considered more of a gimmick rather than serious authentication methods. They are not secure enough and are often not reliable, for instance because they do not work well in dark rooms.

We believe our work shows that face recognition can become a secure and really trustworthy authentication method while being as or even more convenient than existing authentication methods. The tests on our database shows that the probability of accepting an unauthorized person is low enough to secure everything that is not top secret, and face recognition with a depth camera is free from most inconveniences of known bio-authentication methods – such as fingerprint scanners not being usable in gloves or having to look directly at the camera for retina scanners.

Additionally, we showed a proof of concept for skin recognition using multispectral images. We believe that with further research into using higher wavelengths or using more different wavelengths, which we were not able to conduct due to a lack of needed hardware, this could improve the security of face authentication even further.

Bibliography

[1] International Biometrics+Identity Association. Behavioral Biometrcs.

[2] Aleš Procházka, Martin Schätz, Oldřich Vyšata, and Martin Vališ. Microsoft Kinect Visual and Depth Sensors for Breathing and Heart Rate Analysis. URL http://www.mdpi.com/1424-8220/16/7/996/htm.

[3] Tristan Hearn. Webcam pulse detector. URL https://github.com/thearn/webcam-pulse-detector.

[4] Xiaopeng Zhang, Terence Sim, and Xiaoping Miao. Enhancing Photographs with Near Infrared Images. 2008.

[5] Elli Angelopoulou. The Reflectance Spectrum of Human Skin. 1999.

[6] Michael J. Mendenhall, Abel S. Nunez, and Richard K. Martin. Human skin detection in the visible and near infrared. *Appl. Opt.*, 54(35):10559–10570, Dec 2015. doi: 10.1364/AO.54.010559.

[7] Yusuke Kanzawa, Yoshikatsu Kimura, and Takashi Naito. Human skin detection by visible and near-infrared imaging. 2011.

[8] Ed Oswald. With a built-in molecular spectrometer, this phone can identify any material. URL https://www.digitaltrends.com/cool-tech/changhong-smartphone-spectrometer-ces-2017/.

[9] Jose M. Chaves-González, Miguel A. Vega-Rodríguez, and uan M. Sánchez-Pérez Juan A. Gómez-Pulido. Detecting skin in face recognition systems: A colour spaces study. Oct 2009.

[10] futureelectronics.com. What is an Image Sensor? URL http://www.futureelectronics.com/en/sensors/image.aspx.

[11] edmundoptics.com. Imaging Electronics 101: Understanding Camera Sensors for Machine Vision Applications. URL https://www.edmundoptics.com/resources/application-notes/imaging/understanding-camera-sensors-for-machine-vision-applications/.

[12] siliconimaging.com. RGB "Bayer" Color and MicroLenses. URL http://www.siliconimaging.com/RGB%20Bayer.htm.

[13] Tekla Inc. Open camera remote. URL https://play.google.com/store/apps/details?id=net.sourceforge.opencameraremote.

[14] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. 2015.

[15] Rui Min, Neslihan Kose, and Jean-Luc Dugelay. KinectFaceDB: A Kinect Database for Face Recognition. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, 44 (11):1534–1548, Nov 2014. ISSN 2168-2216. doi: 10.1109/TSMC.2014.2331215.

[16] R.I. Høg, P. Jasek, C. Rofidal, K. Nasrollahi, and T.B. Moeslund. An RGB-D Database Using Microsoft's Kinect for Windows for Face Detection. *The IEEE 8th International Conference on Signal Image Technology and Internet Based Systems*, 2012.

[17] Stefano Berretti, Alberto Del Bimbo, and Pietro Pala. Superfaces: A Super-Resolution Model for 3D Faces. In *ECCV Workshops (1)*, pages 73–82, 2012.

[18] OpenKinect. libfreenect. URL `https://github.com/OpenKinect/libfreenect`.

[19] OpenKinect. libfreenect2: Release 0.2. Apr 2016. doi: 10.5281/zenodo.50641.

[20] Adam Geitgey. Face recognition. 2017. URL `https://github.com/ageitgey/face_recognition`.

[21] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection.

[22] Gaurav Goswami, Samarth Bharadwaj, Mayank Vatsa, and Richa Singh. On RGB-D Face Recognition using Kinect.

[23] Alexander Jung. imgaug, 2017. URL `https://github.com/aleju/imgaug`.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. Microsoft Research, 2015.