FaceGuard: A novel e2e face recognition powered two-factor-authorization approach

Janine Marquardt*, Jannik Straube†, Jan-Philipp Watzelt‡ and Jonathan Frick§
Information Systems, TU Munich
Email: *janine.marquardt@tum.de, †jannik.straube@tum.de, ‡jan-philipp.watzelt@tum.de, §jonathan.frick@tum.de

I. Introduction

2-factor authentication methods have become the industry standard for secure authentication. Most often two devices are needed to execute this process. FaceGuard tries to leverage a state-of-the-start face-recognition pipeline with a one-shot-learning approach to enable end to end 2-factor authentication with only one device.

II. FaceGuard Product

A. Features and Functionality

The FaceGuard architecture is secure storage for user data and provides authentification means for third party applications. As seen in figure 1, the application consists of a Python Flask Backend server as well as a React ant design based frontend. FaceGuard allows a user to register with usual email and password and combines this with an image of the user's face. Every time the user then uses FaceGuard to login to other applications a webcame image is used as a second factor.

B. One-Shot training prediction pipeline

At the core of FaceGuard lies a one-shot training and prediction pipeline that makes use of multiple machine learning models. For training, a face-detection and crop model is utilized to detect the bounding box of the user's face. This data is then augmented in the training process. The features of the face are extracted using a faceembedding model. The user credentials are stored in a NoSQL database and the mapping is added to a classifier. The connection is made via the unique user key. When it comes to prediction the process is quite similar. After the bounding box prediction of the face and subsequent feature extraction, a prediction is made using the classification model. If the prediction matches the user credentials that are stored in the database, access is granted.

III. Faceembedding model training

In this chapter, we will illustrate our embedding model creation as this lies at the core of the prediction pipeline. First, we will present the dataset we used and the necessary data preprocessing. Since we have separated our deep learning solution architecture into a face embedding model and a prediction model, we will first discuss the implementation of the embedding model.

Here we will first present our approaches, which we have separated according to two different loss functions. Then we will explain our used machine learning models and give an overview of our hyperparameter tuning and our results. Finally, we will present our prediction solution.

A. Dataset, Face Detection and Data Augmentation

Due to its broad availability and acceptance, we used the Labeled Faces in the Wild dataset (LFW) [2]. This a freely available database of face photographs designed for studying the problem of unconstrained face recognition. The dataset contains more than 13,000 images of faces of 5,749 individuals collected from the web. Each face has been labeled with the name of the person pictured. 1,680 of the people pictured have two or more distinct photos in the dataset.

The first step in face recognition task is the detection and alignment of a person's face in an image. For this, we have used the widely known Multi-task Cascaded Convolutional Network (MTCNN). MTCNN is a python (pip) library written by the Github user "ipacz", which implements the paper "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks" [1]. MTCNN is able to predict a

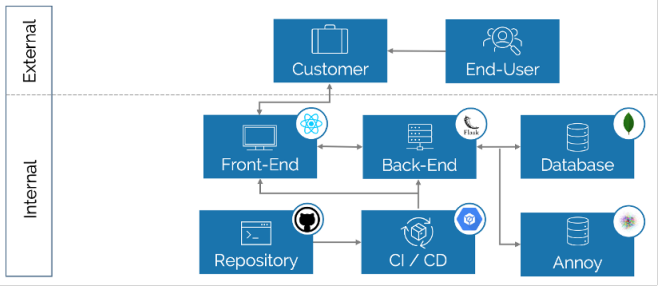

Fig. 1: Solution architecture

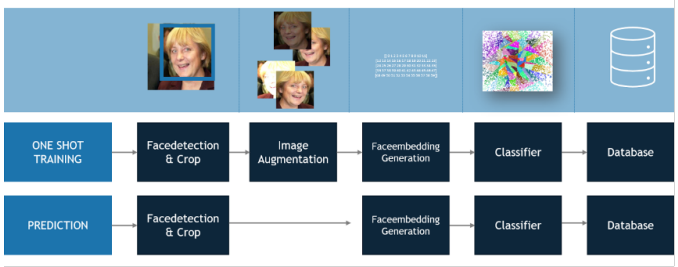

Fig. 2: One-Shot training and prediction pipeline

face and landmark location in an image. We used this model, to detect the faces in the images of the LFW dataset, cropped the persons' face at the bounding box out of the image, and saved it.

As presented in the paper "Face Recognition using One-shot Learning" [3], the accuracy of a face recognition model increases, if the number of training images per person increases. A way to overcome the one-shot learning problem, which means, learning from a single training item, is to use data augmentation. Since we have more than 3,000 individuals in our LFW dataset, who only have one image available, we used basic augmentation techniques, such as rotation and filtering, to enhance our dataset. On top, we also generated a mask augmented dataset, so that we could also train a model, which is able to match a masked face with its corresponding full face, which is an important task especially during the global outbreak of COVID-19.

To generate our advanced dataset, we applied three different data augmentation techniques to each image in the plain LFW dataset. We have made sure that the augmentations are as realistic as possible and that they can actually occur in reality, when a person takes a picture of himself/herself, in order to make our model more robust. Therefore, we have decided to flip the image horizontally and to apply two different brightness changes, where we brighten one image and darken another one. For this task, we have used torchvision.transforms functions.

To generate our masked augmented dataset, we have used the python library "MaskTheFace", based on the Paper "Masked Face Recognition for Secure Authentication" [22]. "MaskTheFace" is a computer vision-based script to mask faces in images. It uses a dlib based face landmarks detector to identify the face tilt and six key features of the face, which are necessary for applying a face mask. Based on the face tilt, that is, whether the person is looking forward, to the right, or to the left, a corresponding mask template is selected from the library of masks. The template mask is then transformed based on the six key features to fit perfectly on the face. As the last step, the brightness of the mask is adjusted to match the images' brightness.

B. Model Training

In this section, we will illustrate the training process of our embedding models. As presented in the paper "Facenet: A unified embedding for face recognition and clustering" [5], which is a face recognition system developed in 2015 by researchers at Google, an essential task to train a face recognition system is to extract high-quality features from faces, called face embeddings. According to this approach, we first trained a model, which is able to extract key figures from an image of a person's face into an embedding vector.

During our training process we did experiments with two different loss functions, the contrastive loss, and the triplet loss, which are both suitable for similarity learning tasks.

For training with the contrastive loss function positive and negative pairs of training, images are used. Positive pairs are composed of an anchor sample and a positive sample, which is similar to the anchor sample, and negative pairs composed by an anchor sample and a negative sample, which is dissimilar to the anchor sample. Since we perform a supervised learning task, the pairs are completed with a corresponding label, e.g., label "1" for positive pairs, and label "0" for negative pairs. The objective is to learn face embedding vector representations with a small distance between them for positive pairs, and a greater distance than some margin value for negative pairs. The contrastive loss forces embedding vectors to have 0 distance for positive pairs, and a distance greater than a margin for negative pairs.

For training with the triplet loss function, we have used triplets of training images, instead of pairs. The triplets are formed by an anchor sample, a positive sample, and a negative sample. The objective is that the distance between the anchor sample and the negative sample embedding representations is greater (and bigger than a margin) than the distance between the anchor and positive embedding representations.

To sum it up, our overall objective is, to minimize the intra-class distances, which means minimizing the distances between the embedding vectors of image of the same person and maximizing the inter-class distances by maximizing the distances between the embedding vectors of images of different persons.

We conducted our experiments with five different kinds of torchvision.models to validate if changes in the model architecture increase our accuracy. Resnet models were proposed in "Deep residual learning for image recognition" [6]. According to the paper, residual networks are easier to optimize and can gain accuracy from considerably increased depth. To determine the best model depth for our learning task, we have conducted experiments with four different Resnet-models, which had 18,34,50, and 101 layers respectively.

Additionally, we trained an Inception-Resnet-v2. This is a variation of the earlier Inception V3 model which borrows some ideas from the previously presented Microsoft ResNets. As presented in "Inception-v4, inception-resnet and the impact of residual connections on learning" [7], the Inception-Resnet-v2 achieves a new state-of-the-art in terms of accuracy on the Large-Scale Visual Recognition Challenge 2012 [8].

C. Model optimization

In order to increase the accuracy of our models, we have not only experimented with different model architectures but also applied hyperparameter tuning. Due to dependency issues and accompanying time issues, we were not able to integrate a framework for automated hyperparameter optimization like ray tune. Therefore, we have followed a manual approach to tune our hyperparameter.

As presented in the FaceNet paper [5], they are using a 128-dimensional embedding output vector. Although they have already stated that changes in the embedding dimensionality do not have significant influences on the accuracy of the model, we still wanted to try out different output dimensions. Since the original output of the Resnet models and the Inception-

Model	Inception ResNet V1 (Reference)	ResNet-18	ResNet-18	ResNet-34	ResNet-34	ResNet-34	ResNet-50
Loss Function	-	Triplet	Contrastive	Triplet	Triplet	Contrastive	Contrastive
Epochs	-	51	60	67	67	60	60
Train Loss	-	0,018	0,000195	0,0095	0,014	0,00014	0,0017
Val Loss	-	0,07	0,0059	0,064	0,035	0,0056	0,0081
Test Acc	93,60%	88,50%	97,70%	91,40%	89,00%	93,40%	89,10%
Test Threshold	1,1	15,4	0,9	16,2	14,5	0,9	0,9
Embedding Size	8631	1000	256	1000	512	256	1000

TABLE I: Experiment results

Resnet-v2 is very large, with a 1000-dimensional embedding, we compared the accuracy of the models trained with the 1000-dimensional output with ones only trained with a 128-d and 256-d output. Our results regarding the accuracy of the models are consistent with the results of the FaceNet paper, which means that we did not find any major differences in accuracy depending on the output size. However, the benefit of a smaller embedding output is a much greater efficiency during the evaluation of the model. Since it was much faster to evaluate the testset on a model with a smaller embedding output, we used a 256-d output for some experiments.
We also tried out different batch sizes, since it is an important hyperparameter that influences the dynamics of the training process. We trained the Resnet-34 with a batch size of 16, 64, and 256. We achieved the lowest validation loss with a batch size of 64 after training the model for 60 epochs.
Another important hyperparameter is the learning rate, which has a huge influence on the performance of a model. It gradually reduces the impact of noise on the network and determines how fast or slow we will move towards the optimal weights. With a too high learning rate, we will skip the optimal solution. If the learning rate is too small, it may never converge or get stuck on a suboptimal solution. We have made experiments with different loss functions, but decided to use a learning rate scheduler as proposed in the paper "How does learning rate decay help modern neural networks?" [9]. They have observed that a learning rate decay helps both optimization and generalization. Therefore, we used a torch.optim learning rate scheduler which adjusts the learning rate every epoch. In detail, we used the ExponentialLR scheduler, which decays the learning rate of each parameter group by gamma every epoch.

IV. FaceEmbedding model Evaluation

A. Performance Measurement approach

Our prediction, whether two images are from the same person or not, is based on a threshold and a distance. We calculate the distance between an anchor embedding and a compare embedding and if the distance is larger than a certain threshold, we predict, we return that the images are not the same (0) otherwise they are the same (1). In order to calculate the best threshold, to achieve the highest accuracy of our model, we used an array of thresholds from 0 to 40, went up with a step-size of 0.1, and calculated the accuracy of the model for each threshold.

B. Result Discussion

In the following, we will discuss the model experiments using the above-mentioned performance measurement approach. To generate a baseline performance measurement the so-called facenet python model is used. This model is based on an Inception Resnet V1 and achieves an LFW accuracy of 99,65%. In this case, the model was trained on the VGGFace2 dataset and the LFW dataset was used for validation. [21] With our measurement, the model achieved an accuracy of 93,60% at a prediction threshold of 1,1.
In our experiments, the Resnet-18 model achieved an accuracy of 88,50% using the triplet loss and 92,7% using contrastive loss. With a larger model size, the Resnet-34 outperformed the former Resnet with an accuracy of 91,4% and 93,4% using tripletloss and contrastive loss. All triplet loss models so far have an output size of 1000, whereas the contrastive loss models use an output size of 256. In all of our experiments, the models with contrastive loss outperformed models trained with triplet-loss. This is also indicated by the level of loss, which is always lower with the former than the latter. Interestingly, the larger model size as stated does not seem to be the reason for best model performance, as a ResNet-50 reaches an accuracy of 89,1% with a higher loss than the ResNet-34 models.
After all the ResNet-34 is chosen for FaceGuard due to its best performance on the test-dataset.

V. FaceEmbedding Prediction

When it comes to the classification task of face embedding vectors the easiest approach would consist of comparing an unseen embedding vector with all existing face embedding vectors in a database. We will refer to this as the memory database approach from now on. Another approach would be to build a classifier based on a support vector machine and train it every time when a new embedding vector is added. A more memory-driven approach could be the use of Spotify's annoy model, which is ideal for comparing embedding vectors, as it is used for song prediction / recommendation. To choose the best classifier for the embedding models an experiment based on training and execution time is setup. 150 training images are used to train each classification approach and one image is used for prediction. The used embeddingvectorsize is 512. Beforehand it was checked that all classifiers are actually

Method	Trainingtime	Predictiontime
Memory Database	0,00026s	0,011s
Annoy	0,016s	0,00067s
SVM	0,96s	0,0034

TABLE II: Training and prediction speed comparison of suitable classification approaches

able to classify this newly added image correctly. For the test, a virtual machine with 60 GB RAM and 16 vCPUs is used. The experiments are fully run on the CPU.
When it comes to training time the memory database obviously performs best as no training is required. The training time of the SVM is measured at 0,98 seconds wheras the annoy model only requires 0,016 seconds.
The most important task for this use-case is the inference time. Here, the memory database approach takes 0,011 seconds, which is slower than the SVM with 0,0034 seconds and the annoy model with 0,00067 seconds.
After all, this underlines the decision to use annoy as the most suitable classifier for classification and verification use-case in FaceGuard.

VI. Main Challenges and Lessons Learnt

The main challenges started with finding publicly available datasets. Most of the formerly publicly available high-quality datasets such as the Vgg2Face set resulted in broken links and references. The Labeled Faces in the Wild dataset remained the most accessible at the time of this writing. Nonetheless, we are quite content with the results we were able to achieve with this dataset. One of our major learnings included the creation of the dataset splits, which are essential for proper model performance measurement. In our first experiments, we were able to achieve extremely high accuracy values e.g. accuracy of 96% after 10 epochs on the test set. After some reflection, it turned out that the train/test split was executed after the pairs/triplets were created leading to seen data in the test set. Therefore it was essential to execute the dataset split on the person-folder level and generating the pairs/triplets afterward. Another major learning included the time spent with system operations tasks including installing proper package versions and solving package mismatches. Also, the manual memory management of the GPU is essential, hence one has to think about the batch size and the resulting GPU consumption. The same applies to running experiments on the GPU: One can not simply loop over a set of thousand images and create embedding vectors.
To summarize - our learnings did not only come from solving machine learning specific tasks, but also solving tasks related to the machine learning environment as well as the deployment of final models.

VII. Outlook

A. Automatic Hyperparameter Tuning

Due to time constraints and dependency issues, we were only able to perform a manual tuning of our hyperparameters. Since hyperparameter tuning is a crucial task in machine learning, as the performance of a model highly depends on the right choice of hyperparameters, it would be one of the most important parts to integrate a tool for automatic hyperparameter optimization. We have already explored different state-of-the-art hyperparameter frameworks, which are applicable for PyTorch Models like Ray Tune [13] and Optuna [14]. When using these frameworks, our focus would be on tuning our learning rate, learning rate decay, batch size and optimizer, in order to increase the accuracy of our model.

B. Modify our models

During our training process, we have used different kinds of torchvision models for feature extraction. In order to increase the generalization of our model, we could modify the models by adding dropout layers. Using dropout is essentially a simple way to prevent a model from overfitting. During training, outputs of the layer will be dropped out by a certain probability (often 0.5), which reduces the co-adaption between neurons, and basically enables the neurons to learn better features [15]. Another idea would be, to design a CNN from scratch.

C. Change dataset

In general, there exist several datasets on the internet for academic purposes, which are suitable for training a model on a face recognition task. However, some of them have been taken off the internet for data protection reasons like for example the MS Celeb dataset from Microsoft, which contained more than 10m images of nearly 100,000 individuals [12]. For this reason, we were only able to train our models with the freely accessible LFW dataset, which only contains 13,233 images of 5,749 individuals [2]. Since the performance of the model is also directly related to the size and quality of the training dataset, it would be an important point to get access to a larger dataset. Moreover, we would be able to ensure a proper split between train, validation, and test set without manual interaction, as many common papers propose using different datasets for those tasks.

D. Evaluate market potential

Since data security is playing an increasingly important role these days and many platforms are already using a two-factor authentication to offer an additional level of security for the accounts of the users, we see a great market potential in our market idea “FaceGuard”. The next step would be now, to conduct detailed market research in order to determine the viability of “FaceGuard” , by discovering the target market, potential competitors, and customers.

VIII. Ethical Aspect of Face Recognition

Nowadays face recognition has become part of everyday life due to Apple’s introduction of Face ID and the subsequent implementation of this feature by the most common mobile phone manufacturers. Millions of people use an AI system daily to unlock their smartphones. That facial recognition can not only be useful to unlock smartphones is obvious. One of

the industrial users of AI is the government of New Zealand. They used it to check whether a passport photo is valid or not. In 2016 this has led to issues because this experimental system has avoidably recognized the eyes of Asians as closed. The root cause of this was that the photos had issues with lighting and that the shown error message has been generic, but this still shows the relevance of discussing ethics in face recognition [13].

The ethical aspects of face recognition are strongly connected with biases. Biases don't need to be conscious; they as well can be unconscious. When processing and interpreting the information the systematic error in thinking may occur and influences decision making. Biases can be related to memory or not paying enough attention and therefore overseeing things [14].

Especially unconscious biases can lead to issues in the development of algorithms. Already in 2003 research of the National Institute of Standards and Technology realized that algorithms having more issues to recognize female subjects than male. In 2018 research of MIT and Microsoft showed, that face identification for white males had an error rate of 1%, but almost 35% for a dark-skinned woman [15].

Those biases are often correlated to the induction theory. Only because every sheep of a herd is white, doesn't mean that the nth+1 sheep is white as well.

Consequently, the quality and diversity of a well-chosen dataset do influence the machine learning algorithm. Realizing this, the EU government published the framework "Ethics guidelines for trustworthy AI" in 2020. One part of this framework is the requirement, that e.g. face recognition systems must reflect "all relevant dimensions of gender, ethnicity and other possible grounds of prohibited discrimination" in their dataset [16]. But not only unconscious biases are dangerous. Also, the conscious misusing of algorithms like in deep fakes can be dangerous [17].

To prevent any kind of discrimination IBM introduced an ethics board for AI [18]. Also, they are working on labs for tech-ethics [19]. But how controversial and difficult ethics in facial recognition is, can best be seen in the stop of offering facial recognition software and further research from e.g. IBM [20].

The future of ethical standards in AI-driven systems will be interesting, especially because there will always remain the question: What is the ethical standard and how is it determined?

References

[1] Zhang, Kaipeng et al. "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks." IEEE Signal Processing Letters 23.10 (2016): 1499–1503. Crossref. Web.

[2] http://vis-www.cs.umass.edu/lfw/ last accessed 10.02.2021

[3] Thakurdesai N, Raut N, Tripathi A. Face Recognition using One-shot Learning. International Journal of Computer Applications. 2018;975:8887.

[4] https://github.com/aqeelanwar/MaskTheFace last accessed 10.02.2021

[5] Schroff F, Kalenichenko D, Philbin J. Facenet: A unified embedding for face recognition and clustering. InProceedings of the IEEE conference on computer vision and pattern recognition 2015 (pp. 815-823)

[6] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778)

[7] Szegedy C, Ioffe S, Vanhoucke V, Alemi A. Inception-v4, inception-resnet and the impact of residual connections on learning. InProceedings of the AAAI Conference on Artificial Intelligence 2017 Feb 12 (Vol. 31, No. 1)

[8] http://image-net.org/challenges/LSVRC/2012/ last accessed 10.02.2021

[9] You K, Long M, Wang J, Jordan MI. How does learning rate decay help modern neural networks?. arXiv preprint arXiv:1908.01878. 2019 Aug 5

[10] Hadsell R, Chopra S, LeCun Y. Dimensionality reduction by learning an invariant mapping. In2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06) 2006 Jun 17 (Vol. 2, pp. 1735-1742). IEEE.

[11] https://synerise.com/papers/supplementary-material-for-fashion-retrieval last accessed 10.02.2021

[12] https://gombru.github.io/2019/04/03/ranking_loss/ last accessed 10.02.2021

[13] Reuters (2016): New Zealand passport robot tells applicant of Asian descent to open eyes, https://www.reuters.com/article/us-newzealand-passport-error-idUSKBN13W0RL last accessed 28.01.2021

[14] Kendra Cherry (2020): What Is Cognitive Bias, https://www.verywellmind.com/what-is-a-cognitive-bias-2794963last accessed 28.01.2021

[15] Larry Hardesty (2018): Study finds gender and skin-type bias in commercial artificial-intelligence systems, https://news.mit.edu/2018/study-finds-gender-skin-type-bias-artificial-intelligence-systems-0212 last accessed 28.01.2021

[16] European Comission (2019): Ethics guidelines fortrustworthy AI, https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai last accessed 10.02.2021

[17] Jonah Peretti, Jared Sosa (2018): http://www.youtube.com/watch?v=cQ54GDm1eL0 last accessed 10.02.2021

[18] Francesca Rossi (2020): How IBM Is Working Toward a Fairer AI, https://hbr.org/2020/11/how-ibm-is-working-toward-a-fairer-ai last accessed 10.02.2021

[19] Patrick Gibbons (2020): Notre Dame, IBM launch Tech Ethics Lab to tackle the ethical implications of technology, https://news.nd.edu/news/notre-dame-ibm-launch-tech-ethics-lab-to-tackle-the-ethical-implications-of-technology/ last accessed 10.02.2021

[20] Jay Peters (2020): IBM will no longer offer, develop, or research facial recognition technology, https://www.theverge.com/2020/6/8/21284683/ibm-no-longer-general-purpose-facial-recognition-analysis-software last accessed 22.01.2021

[21] https://github.com/timesler/facenet-pytorch

[22] Anwar A, Raychowdhury A. Masked Face Recognition for Secure Authentication. arXiv preprint arXiv:2008.11104. 2020