Trustworthy Face Recognition
Improving Generalization of Deep Face Presentation Attack Detection

Thèse n. 7635 2020
présentée le 27 février 2020
à la Faculté des sciences et techniques de l'ingénieur
Laboratoire de l'IDIAP
Programme doctoral en génie électrique
École polytechnique fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Amir Mohammadi

acceptée sur proposition du jury :

Prof. J-Ph. Thiran, président du jury
Prof. Hervé Bourlard, directeur de thèse
Dr. Sébastien Marcel, codirecteur de thèse
Prof. J. Fierrez, rapporteur
Prof. R. Ramachandra, rapporteur
Prof. A. Alahi, rapporteur

Lausanne, EPFL, 2020

Abstract

The extremely high recognition accuracy achieved by modern, convolutional neural network (CNN) based face recognition (FR) systems has contributed significantly to the adoption of such systems in a variety of applications, from mundane activities like unlocking phones to high-security applications such as border-control. Nonetheless, they have been shown to be highly vulnerable to *presentation attacks* (PA), also known as spoof-attacks.

A face PA is said to have occurred when a face biometric-sample is presented to the camera of an FR system with the intention of interfering with the operation of biometric recognition. An example PA is when someone tries to illicitly access an FR system by presenting a printed face photo of an authorized person to the camera. State-of-the-art face presentation attack detection (PAD) systems which are based on CNNs as well offer counter-measures to PAs.

Over the past decade, several datasets have been collected and publicly shared by different research groups, for face PAD experiments. It has been shown that most face PAD systems do not generalize well. That is, PAD systems show satisfactory classification performance when they are trained and evaluated on disjoint subsets of a dataset (known as an intra-dataset evaluation). However, their performance degrades significantly when they are trained using data from one dataset and evaluated using data from another dataset (a cross-dataset evaluation). The poor generalization of PAD systems precludes FR systems from deployment in many real-world applications.

In this thesis, I address generalization issues in face PAD systems in three ways:

1. Although many CNN architectures have been proposed for face PAD, no systematic evaluation of their classification performance has been done before. Here, I evaluate six different CNN architectures on four face PAD datasets in terms of both intra-dataset and cross-dataset performance, and show that patch-based CNN architectures generalize better. Moreover, I propose a novel CNN that analyzes the face images at different scales. This multi-scale analysis allows the proposed CNN to generalize better compared to baseline CNNs.
2. I formulate the low cross-dataset performance of PAD as a domain shift problem and investigate domain adaptation methods as a solution. I propose a novel domain adaptation method based on the hypothesis that some learned filters in CNNs are domain

specific and do not generalize to the other datasets. Pruning these filters leads to higher performance in both intra-dataset and cross-dataset evaluations.

3. I hypothesize that the variability of face images in an FR dataset are nuisance factors in face PAD systems. Based on that, I propose to model the variability of face images in an FR dataset explicitly and induce invariance to these variabilities in the PAD system. The proposed method shows improvements over the baselines in terms of cross-dataset performance.

Extensive experiments on four recent PAD datasets (*Replay-Mobile*, *OULU-NPU*, *SWAN*, and *WMCA*) are conducted to support the claims. Overall, generalization in face PAD systems still remains a challenge and more research effort is needed to address this problem. Finally, this thesis is reproducible as complete implementation of the baselines and the proposed methods are made available freely via the machine-learning library *Bob*.

Keywords: Face Recognition, Presentation Attack Detection, Anti-spoofing, Reproducible Research, Domain Adaptation, Deep Neural Networks

Résumé

La très grande précision obtenue par les systèmes de reconnaissance faciale basés sur des réseaux de neurones convolutifs (Convolutional Neural Network, CNN) a contribué de manière significative à l'adoption de ces systèmes dans une variété d'applications, du déverrouillage de téléphones aux applications de haute sécurité telles que le contrôle des frontières. Néanmoins, ces systèmes s'avèrent être très vulnérables aux attaques de présentation.

Une attaque de présentation se produit lorsqu'un exemple de visage est présenté à un système de reconnaissance dans le but d'interférer avec le fonctionnement de la reconnaissance biométrique. Un exemple typique d'attaque est lorsqu'un individu essaye d'accéder illégalement à un système de reconnaissance de visage en présentant à la caméra une photo de visage d'une personne autorisée. Les systèmes de détection d'attaque de présentation (Presentation Attack Detection, PAD) à la pointe de la technologie, également basés sur des CNN, offrent cependant des contre-mesures aux attaques de présentation.

Au cours de la dernière décennie, plusieurs bases de données de visages pour des expériences de PAD ont été collectées et partagées publiquement par différents groupes de recherche. Il a été démontré que la plupart des systèmes de PAD pour le visage ont des problèmes de généralisation. Ils affichent des performances satisfaisantes lorsqu'ils sont entraînés et évalués sur des sous-ensembles disjoints d'une même base de données (appelé évaluation intra-dataset). Par contre, leurs performances se dégradent considérablement lorsqu'ils sont entraînés à l'aide de données d'un certain ensemble et évalués sur des données provenant d'un autre ensemble (évaluation croisée ou cross-dataset). La mauvaise généralisation de ces systèmes empêche donc leur déploiement à plus grande échelle dans des applications peu contraintes.

Dans cette thèse, j'aborde les problèmes de généralisation des systèmes de PAD pour le visage de trois manières :

1. Bien que de nombreuses architectures CNN aient été proposées pour la PAD faciale, aucune évaluation systématique de leurs performances n'a été effectuée auparavant. Ici, j'évalue six architectures différentes sur quatre ensembles de données en termes de performances intra-dataset et cross-dataset, et montre que les architectures CNN basées sur des informations locales (patchs) généralisent mieux. De plus, je propose

un nouveau CNN qui analyse les images de visage à différentes échelles. Cette analyse multi-échelles permet au CNN proposé de mieux généraliser par rapport aux CNN existants.

2. La faible performance des systèmes de PAD lors d'évaluations croisées est formulée comme un problème de changement de domaine et j'étudie donc les méthodes d'adaptation de domaine comme solution. Je propose une nouvelle méthode d'adaptation de domaine basée sur l'hypothèse que certains filtres appris dans les CNN sont spécifiques à un domaine et ne se généralisent pas aux autres ensembles de données. La suppression de ces filtres entraîne des performances supérieures à la fois dans les évaluations intra-dataset et cross-dataset.
3. J'émets l'hypothèse que la variabilité des images de visage dans un ensemble de données sont des facteurs de nuisance pour les systèmes de PAD. Sur cette base, je propose de modéliser explicitement la variabilité des images de visage dans un ensemble de données dans le but d'introduire une invariance à ces variabilités dans le système de PAD. La méthode proposée montre des améliorations en termes de performances lors d'évaluation croisées.

Des expériences approfondies sur quatre bases de données récentes pour la PAD (*Replay-Mobile*, *OULU-NPU*, *SWAN* et *WMCA*) sont menées pour étayer ces affirmations. Dans l'ensemble, la généralisation des systèmes de PAD pour le visage reste un défi et des efforts de recherche supplémentaires sont nécessaires pour résoudre totalement ce problème. Aussi, cette thèse est reproductible : l'implémentation complète des approches de référence ainsi que les méthodes nouvellement proposées sont mises à disposition librement via la librairie *Bob*.

Keywords : Reconnaissance faciale, Détection d'attaque de présentation, Anti-usurpation, Recherche reproductible, Adaptation de domaine, Réseaux de neurones profonds

Contents

List of Figures

List of Tables

Notation

This section provides a concise reference describing notation used throughout this document. If you are unfamiliar with any of the corresponding mathematical concepts, Goodfellow, Y. Bengio, and Courville (2016) describe most of these ideas in chapters 2–4 of the "Deep Learning" book[1].

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathsf{\mathbf{A}}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0,\dots,0,1,0,\dots,0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

[1]Available freely at https://www.deeplearningbook.org/

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0,1\}$	The set containing 0 and 1
$\{0,1,\dots,n\}$	The set of all integers between 0 and n
$[a,b]$	The real interval including a and b
$(a,b]$	The real interval excluding a but including b
$\mathbb{A}\backslash\mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i,j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i,j,k) of a 3-D tensor $\mathbf{\mathsf{A}}$
$\mathbf{\mathsf{A}}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Linear Algebra Operations

$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	Moore-Penrose pseudoinverse of $\boldsymbol{A}$
$\boldsymbol{A}\odot\boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\det(\boldsymbol{A})$	Determinant of $\boldsymbol{A}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathsf{\mathbf{X}}} y$	Tensor containing derivatives of y with respect to $\mathsf{\mathbf{X}}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$\mathrm{a} \perp \mathrm{b}$	The random variables a and b are independent
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	They are conditionally independent given c
$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E} f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \| Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \rightarrow \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x};\boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\frac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\vert\vert\boldsymbol{x}\vert\vert_p$	L^p norm of $\boldsymbol{x}$
$\vert\vert\boldsymbol{x}\vert\vert$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0,x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise

Sometimes we use a function f whose argument is a scalar but apply it to a vector, matrix, or tensor: $f(\boldsymbol{x})$, $f(\boldsymbol{X})$, or $f(\mathsf{X})$. This denotes the application of f to the array element-wise. For example, if $\mathsf{C} = \sigma(\mathsf{X})$, then $C_{i,j,k} = \sigma(X_{i,j,k})$ for all valid values of i, j and k.

Datasets and Distributions

p_{data}	The data generating distribution
$\hat{p}_{\text{data}}$	The empirical distribution defined by the training set
$\mathbb{X}$	A set of training examples
$\boldsymbol{x}^{(i)}$	The i-th example (input) from a dataset
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\boldsymbol{X}$	The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

1 Introduction

In computer science, **biometrics** refers to the measurements of human characteristics to identify or authenticate a person (Jain, Flynn, and Ross, 2007). Human characteristics that are used in biometrics may be physiological ones – such as face, fingerprint, and DNA – or behavioral ones – such as gait pattern and typing rhythm. A biometric system is a pattern recognition system that captures biometric data from individuals using a sensor, extracts features from the acquired data, and compares the features against a reference (model template) set stored in a database (Jain, Ross, and Prabhakar, 2004). A biometric system usually operates in one of the three modes: **enrollment**, **verification**, or **identification**.

- In the enrollment mode, a person's biometric data is captured, some features are extracted from the data, and the person's identifier (*e.g.,* a name or a personal identification number (PIN)) alongside the features are stored in a database for future use.
- In the verification mode, the user claims an identity by presenting a name or a PIN, his or her biometric data is captured, same features are extracted from the data and are compared to the previously stored features, and the system either accepts the claim or rejects it. In verification, the system answers a question like "Does this biometric data belong to Alice?".
- In the identification mode, the user's biometric data is captured, features are extracted from the data, and is compared to all the templates stored in the database. Then, a matched identity is returned by the biometric system if a match is found. In identification, the system answers a question like "Whose biometric data is this?"

The three modes of a biometric system are illustrated in figure 1.1. Enrollment is always needed in biometric systems. Depending on the application, biometric systems are used either in verification mode or in identification mode.

Face biometrics, also called face recognition (FR), has several advantages compared to biometrics using other human characteristics (Jain, Ross, and Prabhakar, 2004). First, face recognition

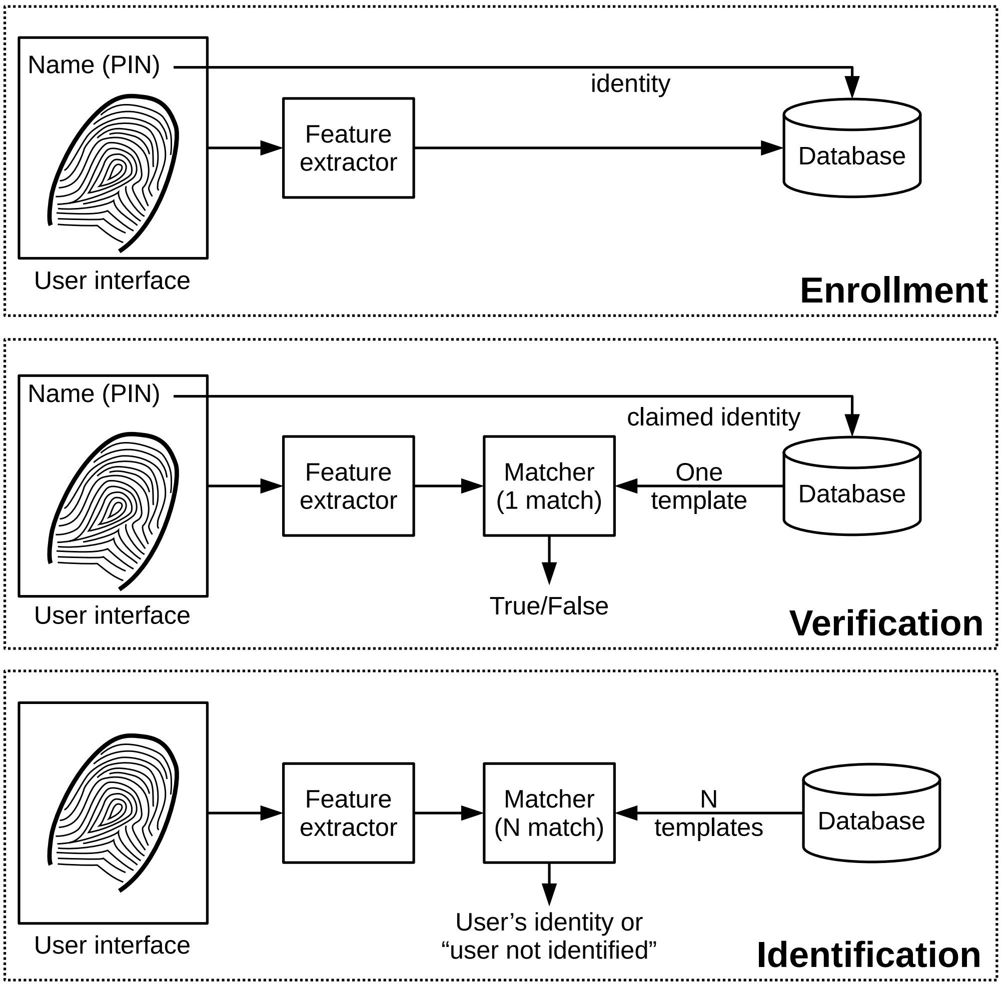


Figure 1.1 – Three operating modes of a biometric system: enrollment, verification, and identification. During enrollment, the user's template (features) is stored. Depending on the application, stored templates are matched against new queries in verification or identification modes. A biometric system is mainly made of four components: the sensor (user interface), feature extractor, matcher, and database. Figure adapted from (Jain, Ross, and Prabhakar, 2004).

is a non-intrusive method and the subject does not come into contact with the equipment. Second, it is natural to humans; humans often use faces to recognize each other. Finally, face biometric data can be captured from a distance, potentially without the consent of the people, which allows face biometrics to be used in surveillance applications. Although, this use-case also brings certain privacy concerns.

Face recognition as a research field has evolved over the last 30 years. The first large scale appearance based FR methods such as Eigenfaces (Turk and Pentland, 1991) and Fisherfaces (Wiskott et al., 1997) modeled face images in linear subspaces. Those methods had an acceptable classification performance only under strongly constrained settings. Current state-of-the-art FR methods use deep convolutional neural networks (CNNs) (Taigman et al., 2014; Parkhi, Vedaldi, and Zisserman, 2015; Schroff, Kalenichenko, and Philbin, 2015; Y. Sun et al., 2015). These methods achieve near-perfect recognition accuracy in unconstrained datasets such as ‘labeled faces in the wild’ (LFW) (G. B. Huang et al., 2007). The extremely high recognition accuracy of FR systems has allowed the FR systems to be used in a variety of applications. These applications range from low risk authentication activities such as unlocking phones to high-security applications like border-control. Figure 1.2 includes a promotional image of FR showcasing one of the applications of FR.

Figure 1.2 – An advertisement image promoting availability of FR in a smartphone. The FR system is used for authentication here. The FR system allows the users to unlock their smartphone by presenting their faces to the camera of the phone. Image courtesy of “GEEK KAZU“ from flickr: https://flic.kr/p/2aKDN2c, CC-BY-2.0

However, face recognition systems are highly vulnerable to *presentation attacks* (PA) (Duc and Minh, 2009; Kose and Dugelay, 2013; Hadid, 2014; Mohammadi, Bhattacharjee, and Marcel, 2017). Presentation attacks are biometric samples that are presented to the sensor of the biometric system with the intention of interfering with the intended operation of the biometric system (ISO/IEC DIS 30107-1, 2016). For example, a presentation attack happens when an adversary claims someone else’s identity and presents a printed face photo of the claimed identity to the camera of the biometric system. Samples that are not PAs are called *bona fide* samples. In face biometrics, several types of presentation attacks exist (Marcel et al., 2019) such as:

- print attack: a printed face photo
- photo replay attack: a face photo shown using a digital display
- video replay attack: a face video is shown using a digital display
- mask attack: an artificial mask imitating a person’s face.

Figure 1.3 illustrates some examples of *bona fide* and PA face images. PAs are performed by presenting a *presentation attack instrument* (PAI) (*e.g.,* a printed photo) to the sensor of the biometric system. The process of creating a PAI introduces some artifacts in the captured face image. Face presentation attack detection (PAD) systems use cues based on different criteria to detect presentation attacks. These cues include motion (eye blinking (G. Pan, L. Sun, and Z. Wu, 2008), small involuntary movements of parts of face and head (Anjos and Marcel, 2011; Jee, Jung, and Yoo, 2006)), surface texture of the skin (Chingovska, Anjos, and Marcel, 2012), image quality metrics (Galbally, Marcel, and Fierrez, 2014; Wen, H. Han, and Jain, 2015), near infrared or thermal images (Z. Zhang et al., 2011), and 3D information (George et al., 2019). Face PAD systems that use only a single face image from the visual spectra to detect PAs are more popular because they can be easily integrated into current FR systems.

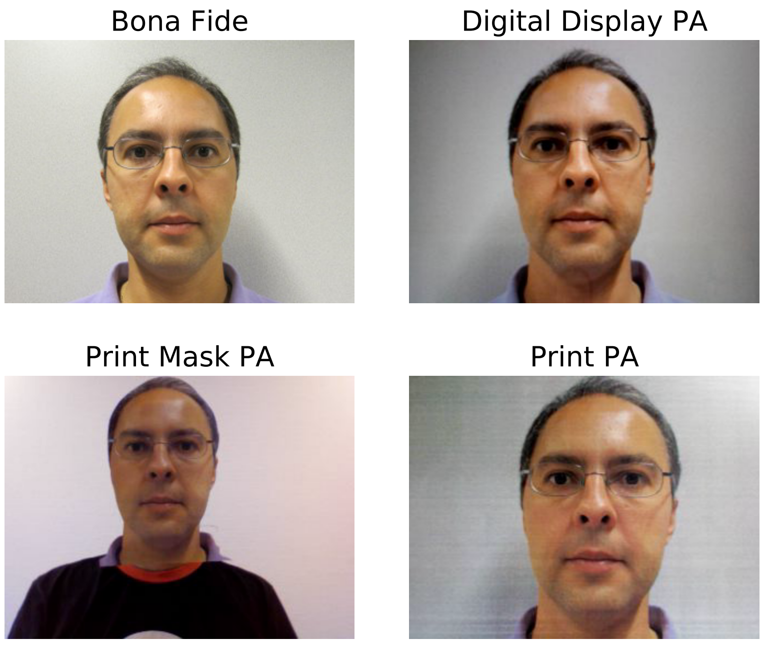


Figure 1.3 – Example face images captured by the camera module of an FR system. The top-left image is a *bona fide* sample. The rest of the images are PAs. Presentation attack detection systems rely on artifacts present in PA samples to detect PAs.

Usually face PAD systems are also machine-learning based, trained using data-driven approaches. Over the past decade, several datasets have been collected and publicly shared by different research groups, for face PAD experiments. These datasets include protocols defining mutually disjoint data subsets for **training**, **development**, and **evaluation** of face PAD methods. After training, two scenarios are possible for evaluating a face PAD system:

- **intra-dataset** evaluation: the evaluation protocol is taken from the same dataset as the training protocol, or,
- **cross-dataset** evaluation: the evaluation protocol is taken from a different dataset than the training protocol.

State-of-the-art face PAD systems, especially those based on CNNs, show promising performance in detecting PAs in intra-dataset evaluation scenarios. Typically, however, their performance degrades significantly when tested in cross-dataset scenarios[1]. In other words, most current face PAD systems do not generalize well. This precludes FR systems from deployment in many real-world applications.

The generalization issues arise because sufficient variation of *nuisance factors* does not exist in the training dataset. In the context of a face PAD system, nuisance factors are identities, illumination, pose, and many other factors. In other words, a PAD system should be capable of reliably detecting presentation attacks without overfitting to certain identities, illumination, pose, and other nuisance factors. Variations of the *nuisance factors* present in face images cause distribution shift between the training and evaluation data.

Most machine learning models work under the assumption that the distribution of data does not change between training and evaluation data. Therefore, a change in the distribution of data, referred to as *domain shift*, can lead to poor performance of the model on the evaluation data (S. J. Pan and Q. Yang, 2009; Quionero-Candela et al., 2009). The low performance of face PAD in cross-dataset evaluations can be seen as a problem of domain shift. In other words, the source (training) dataset can be seen as one domain and target (evaluation) dataset can be seen as another domain.

Methods to compensate for domain shift fall into two categories: *domain adaptation* and *domain generalization* methods. In domain adaptation, we assume that some training data from the target domain is available. Domain generalization methods, by contrast, do not rely on any training data from the target domain. Instead, those methods assume that data from multiple (source) domains are available.

Most domain adaptation methods assume that training data from *all* classes is available in the target domain (Wang and Deng, 2018). However, in face PAD, collecting PAs is more expensive than collecting *bona fide* samples. Moreover, in real-world scenarios, we may always expect a PAD system to be exposed to previously unseen attacks. That is, either the attack type may be different from those represented in the training data or the PAI used to create the attack may be different from those used during training. For example, for a face PAD model that is trained on print and replay attacks, a mask attack may be considered as an unseen attack type. Therefore, domain adaptation methods that can work with only *bona fide* samples from the target domain are desired.

Overall, collecting training data in the target domain is often difficult or impossible because we may not have the opportunity or the resources to collect training data in the target domain. Therefore, domain generalization methods are more appealing compared to domain adaptation methods because they do not use training data from the target domain. As discussed

[1]Note that CNN-based face PAD systems outperform PAD systems that use hand-crafted features in both intra-dataset and cross-dataset evaluations (J. Yang, Lei, and S. Z. Li, 2014; K. Patel, Hu Han, and Jain, 2016; Z. Boulkenafet et al., 2017; Atoum et al., 2017; George and Marcel, 2019)

before, domain generalization methods require training data from multiple (source) domains instead. However, collecting data in multiple domains is difficult as well. To collect data in multiple domains, we must first identify the nuisance factors of data. Variations of nuisance factors is the cause of domain shift between datasets. In face PAD datasets, domain-shift may be caused by a variety of nuisance factors, including: the camera device, resolution of images, distance of the subject to the camera, the instrument used to create the attack, lighting conditions, identity, pose, age, and facial-makeup. Sometimes, a nuisance factor is known explicitly and is categorized in a dataset, that is each sample is labeled by the specific category of the nuisance factor. In such cases, several methods can be used to induce invariance to the nuisance factor in a data-driven model (Y. Bengio, Courville, and Vincent, 2013; Xie et al., 2017; Louizos et al., 2015; Ganin et al., 2016; Y. Li, Swersky, and Zemel, 2014). These methods help data-driven models to learn features that are invariant to the nuisance factor.

However, identifying and also categorizing all the nuisance factors is impossible (Y. Bengio, Courville, and Vincent, 2013). Therefore, it is desirable to use methods that do not rely on explicit categorization to induce invariance to nuisance factors (*e.g.,* (Jaiswal et al., 2018)). Moreover, because most invariance induction methods are data-driven, such as (Ganin et al., 2016; Jaiswal et al., 2018), they require a training dataset with adequate variation of nuisance factors. For example, if the camera device is a nuisance factor, the more different camera models that are used in the data collection, the better the invariance induction method will perform.

Current face PAD datasets do not contain sufficient variation in terms of nuisance factors. For example, currently available face PAD datasets are collected with less than 10 camera devices, with only 50 to 150 identities participating, and have limited variations in lighting conditions. Modern FR datasets, on the other hand, consist of millions of images collected from various sources (Guo et al., 2016). Sufficient variations of nuisance factors are inherently represented in such datasets.

1.1 Contributions

I address the low classification performance of face PAD in cross-dataset scenarios in this thesis.

Many CNN architectures have been proposed for face PAD. However, no systematic comparison of the CNN architectures has been done before. In this work, I systematically evaluate several CNN architectures for face PAD in terms of classification performance in both intra-dataset and cross-dataset scenarios. I show that face PAD CNNs that classify patches of faces images (a part of the face image) independently outperform CNNs that classify the whole face image. This is expected as in face PAD, we are mainly interested in the artifacts presents in all parts of the face image. Classifying the whole face image using a CNN can lead to overfitting. Moreover, I also propose a novel multi-scale CNN that classifies patches of face images at different scales. The proposed architecture shows promising generalization capabilities.

The low classification performance of face PAD in cross-dataset scenarios can be formulated as a domain shift problem. I hypothesize that some learned filters in face PAD CNNs are domain specific that is those filters are more sensitive to domain shift. Based on this hypothesis, I propose to prune the domain specific filters in CNNs to achieve higher classification performance on the target domain. Pruning CNNs also brings the advantage of lowering the computation cost of CNNs. Only *bona fide* samples of the target domain are used to identify domain specific filters in a CNN. This was done based on the fact that collecting PA samples in the target domain is more expensive. Moreover, because the proposed method only uses *bona fide* samples, I investigate a scenario where *bona fide* samples of an FR dataset, instead of *bona fide* samples of the target domain, is used for identifying domain specific filters. Thus, the proposed method can be implemented as both domain adaptation and domain generalization methods depending on which data is used for identifying domain specific filters.

Finally, I propose a method that takes advantage of the large variation of FR datasets to improve the cross-dataset performance of face PAD. I hypothesize that *all* the underlying factors that explain the data in an FR dataset (which contains only *bona fide* samples) are nuisance factors for face PAD. Then, I propose to model the variation of FR samples in an unsupervised manner explicitly. Unsupervised modeling of the underlying factors of data has the advantage that it does not require us to identify and label factors of the FR dataset. Moreover, by modeling these factors explicitly we can induce invariance to these factors in face PAD systems.

Extensive experiments on four recent PAD datasets (*Replay-Mobile*, *OULU-NPU*, *SWAN*, and *WMCA*) are conducted to support the claims. Moreover, the software implementation of the experiments done in this thesis are made available freely to support the future development of this field[2]. The outline of this thesis is described below.

1.2 Outline

Background material required to understand the work in this thesis and related work are given in chapter 2.

- Section 2.1 gives a concise reference to elements of CNNs. This section helps you understand the terminology of CNNs and interpret the different CNN architectures that are illustrated as tables in this thesis.
- Section 2.2 introduces prominent CNN architectures that are relevant in this thesis.
- Section 2.3 overviews state-of-the-art FR systems.
- Section 2.4 outlines state-of-the-art face PAD systems.
- Section 2.5 establishes the terminology of domain adaptation and presents related domain adaptation methods.

[2]https://gitlab.idiap.ch/bob/bob.thesis.amohammadi/

- Finally, sections 2.6 and 2.7 introduce the evaluation metrics and datasets used in this thesis, respectively.

Chapter 3 demonstrates the vulnerability of state-of-the-art FR systems to presentation attacks. We shall see that state-of-the-art FR systems are vulnerable to more than 90% of PAs. It is recommended to read sections 2.3, 2.6 and 2.7 before reading this chapter.

In chapter 4, several CNN architectures are systematically evaluated for face PAD and their performance in both intra-dataset and cross-dataset evaluation scenarios is reported. Moreover, a novel CNN architecture is proposed for face PAD which analyzes face images in multiple scales jointly. It is recommended to read sections 2.1, 2.2, 2.4, 2.6 and 2.7 before reading this chapter.

The problem of low cross-dataset performance of face PAD is formulated as a domain shift problem in chapter 5, and domain adaptation methods are investigated. Moreover, a novel domain adaptation method that uses only *bona fide* samples from the target domain is proposed in chapter 5. It is recommended to read sections 2.4, 2.5, 2.6 and 2.7 before reading this chapter.

In chapter 6, a novel domain generalization method is proposed which models nuisance factors of face PAD in an unsupervised manner using FR datasets. Then, the method induces invariance to the nuisance factors in face PAD models. It is recommended to read sections 2.2.9, 2.4, 2.5, 2.6 and 2.7 before reading this chapter.

Finally, the thesis is summarized in chapter 7 and possible future directions are discussed in section 7.1. Moreover, my contributions in terms of publications and software are listed in detail in sections 7.2 and 7.3, respectively.

2 Background and Related Work

2.1 Background on Convolutional Neural Networks (CNNs)

CNNs (LeCun et al., 1989)[1] were proposed in 1989. However, their adoption increased significantly when Krizhevsky, Sutskever, and G. E. Hinton (2012) won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 image classification challenge (Russakovsky et al., 2015). In this section, I will present the background information on CNNs necessary for the rest of the thesis. Because the readers are assumed to be familiar with deep learning and CNNs, I will present the CNN architectures as tables using the Keras terminology (https://keras.io/) in this thesis. For example, a CNN architecture may be shown as in table 2.1. The CNN components used in tables to detail the architectures are explained below.

Table 2.1 – An example CNN architecture. The terminology of Keras (https://keras.io/) is used for detailing the architecture.

Layer (type)	Details	Output Shape	Parameters
C1 (Conv2D)	filters=6, kernel_size=(5, 5), strides=(1, 1), padding=valid, activation=tanh	(28, 28, 6)	156
S2 (AveragePooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(14, 14, 6)	0
C3 (Conv2D)	filters=16, kernel_size=(5, 5), strides=(1, 1), padding=valid, activation=tanh	(10, 10, 16)	2,416
S4 (AveragePooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(5, 5, 16)	0
C5 (Conv2D)	filters=120, kernel_size=(5, 5), strides=(1, 1), padding=valid, activation=tanh	(1, 1, 120)	48,120
FLATTEN (Flatten)		(120)	0
F6 (Dense)	units=84, activation=tanh	(84)	10,164
OUTPUT (Dense)	units=10, activation=sigmoid	(10)	850
Model	Parameters: total=61,706, trainable=61,706		

2.1.1 Dense Layer

A **Dense** or **fully-connected** layer, is a non-linear transformation layer. Given an input, $\boldsymbol{x}$, which is a d dimensional vector, its output, $\boldsymbol{y}$, is:

$$\boldsymbol{y} = \mathrm{g}(\mathbf{W}\boldsymbol{x} + \mathbf{b}) \tag{2.1}$$

[1]Readers not familiar with CNNs and deep learning are advised to read (Goodfellow, Y. Bengio, and Courville, 2016) available on https://www.deeplearningbook.org/.

where **W** is a $u \times d$ dimensional weight (also called kernel) matrix, **b** is a u dimensional bias vector, and g is a non-linear **activation** function. In the tables of this thesis, a Dense layer is detailed with the parameters shown below:

Dense Parameters	**Explanation**
units	u, one of the dimensions of **W**
activation	name of the activation function

The following non-linear activation functions are mentioned in this thesis:

- **tanh**: the hyperbolic tangent function:
$$y = tanh(x) \tag{2.2}$$
- **sigmoid**:
$$y = 1/(1 + exp(-x)) \tag{2.3}$$
- **softmax**[2]:
$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathcal{R}^K \tag{2.4}$$
  where given K numbers, $(z_1, \ldots, z_K)$, $\sigma(\mathbf{z})_i$ is proportional to the exponentials of z_i. In other words, it normalizes the numbers that are not necessarily between 0 and 1 and do not sum to 1, to numbers between 0 and 1 which sum to 1. These numbers can be interpreted as probabilities.
- **Rectified Linear Unit (ReLU)**:
$$y = max(0, x) \tag{2.5}$$

Figure 2.1 demonstrates the tanh, sigmoid, and ReLU activation functions.

2.1.2 Conv2D Layer

A convolutional layer, is the main building block of CNNs. The 2D term in its name signifies that the input to the layer are two dimensional signals like images. Assume an image, $\mathbf{I}$, as input to a convolutional layer. $\mathbf{I}$ has the size of $H \times W \times C$, where H is its height, W is its width, and C is the number of **channels** (*e.g.,* 3 for RGB images). Then, output of the convolutional layer, $\mathbf{Y}$, commonly referred as **feature maps**, is:

$$\mathbf{Y}(i, j) = \mathbf{b} + \sum_{m=1}^{M} \sum_{n=1}^{N} \mathbf{I}(i \times s_h + m, j \times s_w + n)\mathbf{K}(m, n) \tag{2.6}$$

where $\mathbf{K}$ is an $M \times N \times C \times D$ dimensional weight matrix (the kernel), **b** is a D dimensional bias vector, M and N are the kernel sizes, D is the number of filters in the convolutional

[2]https://en.wikipedia.org/wiki/Softmax_function

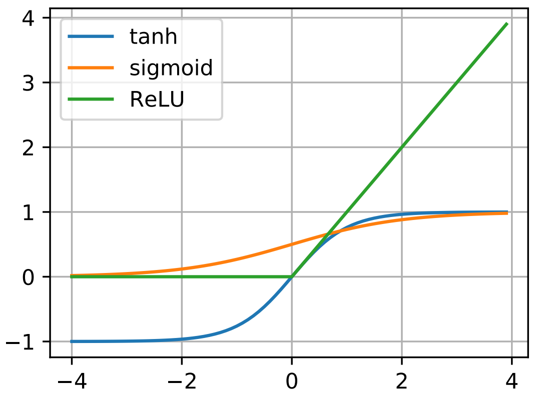


Figure 2.1 – Plot of the tanh, sigmoid, and ReLU activation functions. Output values of the activation functions given different input values is shown.

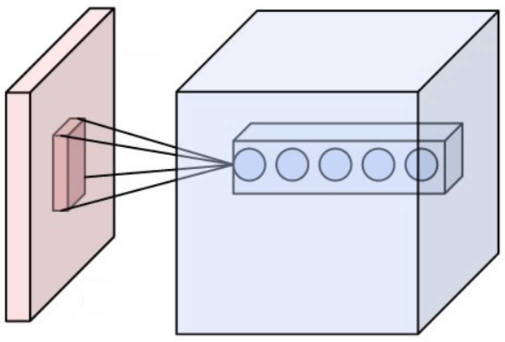


Figure 2.2 – A convolutional layer. A convolutional layer outputs a feature map (blue) by calculating the same kernel operation on local regions of the input (red) and shifting the kernel to cover all the regions of the input. Figure from https://en.wikipedia.org/wiki/Convolutional_neural_network (CC BY-SA 4.0).

layer, s_h and s_w are the strides, and $\mathbf{Y}$ has a dimension slightly smaller in height and width than $\frac{H}{s_h} \times \frac{W}{s_w} \times D$. $\mathbf{Y}$ is smaller than that size because the convolution operation (actually cross-correlation) in equation (2.6) is not valid when $i \times s_h + m$ or $j \times s_w + n$ indexes get bigger than the height and width of $\mathbf{I}$. The convolutional layers typically allow zero padding of the input so that the output has the desired $\frac{H}{s_h} \times \frac{W}{s_w} \times D$ size. Schematics of a convolutional layer is shown in figure 2.2. In the tables of this thesis, a Conv2D layer is detailed with the parameters shown below:

Conv2D Parameters	Explanation
filters	D, one of the dimensions of $\mathbf{K}$
kernel	two numbers which refer to M and N
strides	two numbers which refer to s_h and s_w
padding	either *valid* or *same*. If *valid*, only valid operations are calculated and the output is slightly smaller in height and width than $\frac{H}{s_h} \times \frac{W}{s_w} \times D$. If *same*, the output has the size of $\frac{H}{s_h} \times \frac{W}{s_w} \times D$.
activation	name of the activation function

2.1.3 Conv2DTranspose Layer

Given that a convolution is a linear operation, it can be written as a matrix multiplication (Goodfellow, Y. Bengio, and Courville, 2016, page 356) where the input is reshaped into a vector and the kernel is represented in a sparse matrix. If we transpose the matrix before multiplying it by input, we achieve the *transposed convolution* operation. This operation is needed if we wish to process and upsample the input image. The upsampling is achieved by choosing a stride value higher than 1. A Conv2DTranspose layer will be detailed with the same parameters as the Conv2D layer in this thesis.

2.1.4 ZeroPadding2D Layer

As we saw in convolutional layers, sometimes it is desired to zero pad the input. This padding is either done by the convolutional layer when the *padding* parameter is set to *same* or this can be done in a different layer using the ZeroPadding2D layer. In the tables of this thesis, a ZeroPadding2D layer is detailed with the parameters shown below:

ZeroPadding2D Parameters	Explanation
padding	four numbers that correspond to the number of zero pixels that will be added to the top, bottom, left, and right of the image, respectively.

2.1.5 Cropping2D Layer

A Cropping2D layer, as the name suggests, crops the input image. In the tables of this thesis, a Cropping2D layer is detailed with the parameters shown below:

Cropping2D Parameters	Explanation
cropping	four numbers that correspond to the number of pixels that will be cropped out from the top, bottom, left, and right of the image, respectively.

2.1.6 Pooling2D Layer

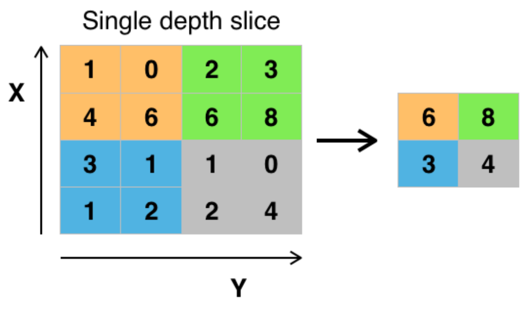


Figure 2.3 – A (maximum) pooling operation. A pooling layer computes either a max, such as in this image, or an average operation on local regions of input. The output is computed by shifting the max or average kernel operation to cover all the regions of the input. Figure from https://en.wikipedia.org/wiki/Convolutional_neural_network (CC BY-SA 4.0).

The pooling layers allow down-sampling of feature maps. In CNNs, typically, the number of channels in feature maps increases as the input is further processed in deeper layers of the network. Pooling layers are used to reduce the spatial dimensions of feature maps which in turn reduces the number of parameters and computations in the network. Using the notation that we used for convolutional layers, the output of a pooling layer is:

$$\mathbf{Y}(i,j) = \mathrm{g}(\mathbf{I}(i \times s_h + 0 \dots i \times s_h + M - 1, j \times s_w + 0 \dots j \times s_w + N - 1)) \tag{2.7}$$

where g is the pooling operation that is typically either the max or the $average$ operation. In simple words, g takes as input a local patch of $\mathbf{I}$ which has the size of $M \times N$ and outputs one value for that patch. This operation is done for each channel dimension, C, separately. The output of a pooling layer, $\mathbf{Y}$, has the size of $\frac{H}{s_h} \times \frac{W}{s_w} \times C$ or slightly smaller in height and width similar to convolutional layers. Schematics of a pooling layer is shown in figure 2.3. In the tables of this thesis, a Pooling2D layer is detailed with the parameters shown below:

Pooling2D Parameters	**Explanation**
Max or Average	specifies g
pool size	two numbers which refer to M and N
strides	two numbers and refers to s_h and s_w. Usually strides has the same value as the pool size.
padding	either *valid* or *same*. If *valid*, only valid operations are calculated and the output is slightly smaller in height and width than $\frac{H}{s_h} \times \frac{W}{s_w} \times C$. If *same*, the output has the size of $\frac{H}{s_h} \times \frac{W}{s_w} \times C$.

2.1.7 GlobalPooling2D Layer

A global pooling layer is an extension of pooling layers where its goal is to reduce the height and width of feature maps to 1. The input to the g function is the full feature map so the max or average operation is applied on all spatial dimensions. If the input has the size of $H \times W \times C$, the output will be a C dimensional vector. Global pooling layers are typically used after the final convolutional layer in a network (Lin, Chen, and Yan, 2013).

2.1.8 Flatten Layer

The flatten layers, simply flatten a matrix to a vector. Given an input matrix of size $H \times W \times C$, the output will be a vector with size $H.W.C$.

2.1.9 Dropout Layer

Dropout introduced by G. E. Hinton et al. (2012) and Srivastava et al. (2014), randomly drops a percentage, referred as drop rate, of neurons, *during training*, in a layer by multiplying its output by 0. The remaining neurons are scaled by $\frac{1}{1-\text{drop rate}}$ so that the total sum of the output does not change. At inference time, a Dropout layer has no effect. Dropout allows us to train large networks while avoiding overfitting. Dropout may be viewed as an approximation of bagging (Goodfellow, Y. Bengio, and Courville, 2016, p.255) where it allows us to train an ensemble of networks which share weights with each other. The percentage of neurons that are dropped during training is parameterized as *drop rate* in the tables of this thesis.

2.2 Relevant CNN Architectures

In the following, I will present some important milestones of deep CNNs that is relevant to this thesis.

2.2.1 LeNet-5

LeNet-5 (LeCun et al., 1998) was one of the first deep CNNs that had been applied successfully on handwriting recognition. A simplified version of its architecture is shown in table 2.2. The network consisted of three convolutional layers and two fully-connected layers. The first two convolutional layers were followed by sub-sampling layers (LeCun et al., 1998) (which I have replaced by average pooling layers in table 2.2). The network had 60,000 parameters.

2.2.2 AlexNet

Krizhevsky, Sutskever, and G. E. Hinton (2012) introduced AlexNet which was a deep CNN. They combined several techniques to improve the training time and accuracy of the CNN significantly and won the (ILSVRC) 2012 image classification challenge (Russakovsky et al., 2015) with a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the second place winner. The network consisted of eight layers where the first five layers were convolutional layers and the last three were fully-connected layers. The authors argued that

Table 2.2 – Architecture details of a simplified version of LeNet-5 (LeCun et al., 1998). The input to this architecture are gray-scale 32 × 32 pixel images of digits.

Layer (type)	Details	Output Shape	Parameters
C1 (Conv2D)	filters=6, kernel_size=(5, 5), strides=(1, 1), padding=valid, activation=tanh	(28, 28, 6)	156
S2 (AveragePooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(14, 14, 6)	0
C3 (Conv2D)	filters=16, kernel_size=(5, 5), strides=(1, 1), padding=valid, activation=tanh	(10, 10, 16)	2,416
S4 (AveragePooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(5, 5, 16)	0
C5 (Conv2D)	filters=120, kernel_size=(5, 5), strides=(1, 1), padding=valid, activation=tanh	(1, 1, 120)	48,120
FLATTEN (Flatten)		(120)	0
F6 (Dense)	units=84, activation=tanh	(84)	10,164
OUTPUT (Dense)	units=10, activation=sigmoid	(10)	850
Model	Parameters: total=61,706, trainable=61,706		

the depth of the network, using five convolutional layers, was essential to the performance of the network. Also, another important change compared to traditional CNNs was using rectified linear units (ReLU) (Sanger, 1989; Nair and G. E. Hinton, 2010), f$(x) = max(0, x)$, as activation functions compared to the hyperbolic tangent (tanh), f$(x) = tanh(x)$, activation function. The tanh activation function has saturating nonlinearities which slows down the gradient descent training. The ReLU activation function does not have this property (see figure 2.1 on page 11). To avoid overfitting of the network, dropout (G. E. Hinton et al., 2012) and data augmentation (horizontal image mirroring, random translation, also known as random cropping, and random RGB color shift Krizhevsky, Sutskever, and G. E. Hinton, 2012) were used. The details of a simplified version of AlexNet is shown in table 2.3. The original network had 60 million parameters which is a 1000 times more than LeNet-5.

Table 2.3 – Architecture details of a simplified version of AlexNet (Krizhevsky, Sutskever, and G. E. Hinton, 2012). The input to this architecture are color 227 × 227 pixel images.

Layer (type)	Details	Output Shape	Number of Parameters
C1 (Conv2D)	filters=96, kernel_size=(11, 11), strides=(4, 4), padding=valid, activation=relu	(55, 55, 96)	34,944
P1 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(27, 27, 96)	0
C2 (Conv2D)	filters=256, kernel_size=(5, 5), strides=(1, 1), padding=same, activation=relu	(27, 27, 256)	614,656
P2 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(13, 13, 256)	0
C3 (Conv2D)	filters=384, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(13, 13, 384)	885,120
C4 (Conv2D)	filters=384, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(13, 13, 384)	1,327,488
C5 (Conv2D)	filters=256, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(13, 13, 256)	884,992
P5 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(6, 6, 256)	0
FLATTEN (Flatten)		(9216)	0
D6 (Dropout)	drop_rate=0.5	(9216)	0
F6 (Dense)	units=4096, activation=relu	(4096)	37,752,832
D7 (Dropout)	drop_rate=0.5	(4096)	0
F7 (Dense)	units=4096, activation=relu	(4096)	16,781,312
OUTPUT (Dense)	units=1000, activation=softmax	(1000)	4,097,000
Model	Parameters: total=62,378,344, trainable=62,378,344		

2.2.3 VGG Networks

Simonyan and Zisserman (2014) investigated the effect of depth on the performance of CNNs in large-scale image classification. They fixed the parameters of convolutional layers and only increased the depth of the network by adding more convolutional layers. They showed that the accuracy of the CNNs can be increased consistently with added depth while keeping the network architecture simple and using only small 3 × 3 convolutional kernels. The test was done on several CNN configurations ranging from 11 to 19 layers. Between configurations, the number of max pooling layers and fully-connected layers were kept constant and only the number of convolutional layers were changed. For example, details of VGG16, which has 16 convolutional and fully-connected layers, and VGG19 are shown in tables 2.4 and 2.5. VGG19, compared to VGG16, has these extra layers: *block3_conv4*, *block4_conv4*, *block5_conv4*.

Table 2.4 – Architecture details of VGG16, configuration D in (Simonyan and Zisserman, 2014). The input to the architecture are color 224 × 224 pixel images.

Layer (type)	Details	Output Shape	Number of Parameters
block1_conv1 (Conv2D)	filters=64, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(224, 224, 64)	1,792
block1_conv2 (Conv2D)	filters=64, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(224, 224, 64)	36,928
block1_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(112, 112, 64)	0
block2_conv1 (Conv2D)	filters=128, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(112, 112, 128)	73,856
block2_conv2 (Conv2D)	filters=128, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(112, 112, 128)	147,584
block2_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(56, 56, 128)	0
block3_conv1 (Conv2D)	filters=256, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(56, 56, 256)	295,168
block3_conv2 (Conv2D)	filters=256, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(56, 56, 256)	590,080
block3_conv3 (Conv2D)	filters=256, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(56, 56, 256)	590,080
block3_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(28, 28, 256)	0
block4_conv1 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(14, 14, 512)	0
block5_conv1 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(7, 7, 512)	0
flatten (Flatten)		(25088)	0
fc1 (Dense)	units=4096, activation=relu	(4096)	102,764,544
fc2 (Dense)	units=4096, activation=relu	(4096)	16,781,312
predictions (Dense)	units=1000, activation=softmax	(1000)	4,097,000
Model	Parameters: total=138,357,544, trainable=138,357,544		

Table 2.5 – Architecture details of VGG19, configuration E in (Simonyan and Zisserman, 2014). The input to the architecture are color 224 × 224 pixel images.

Layer (type)	Details	Output Shape	Number of Parameters
block1_conv1 (Conv2D)	filters=64, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(224, 224, 64)	1,792
block1_conv2 (Conv2D)	filters=64, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(224, 224, 64)	36,928
block1_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(112, 112, 64)	0
block2_conv1 (Conv2D)	filters=128, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(112, 112, 128)	73,856
block2_conv2 (Conv2D)	filters=128, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(112, 112, 128)	147,584
block2_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(56, 56, 128)	0
block3_conv1 (Conv2D)	filters=256, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(56, 56, 256)	295,168
block3_conv2 (Conv2D)	filters=256, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(56, 56, 256)	590,080
block3_conv3 (Conv2D)	filters=256, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(56, 56, 256)	590,080
block3_conv4 (Conv2D)	filters=256, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(56, 56, 256)	590,080
block3_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(28, 28, 256)	0
block4_conv1 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(28, 28, 512)	2,359,808
block4_conv4 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(14, 14, 512)	0
block5_conv1 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(14, 14, 512)	2,359,808
block5_conv4 (Conv2D)	filters=512, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(7, 7, 512)	0
flatten (Flatten)		(25088)	0
fc1 (Dense)	units=4096, activation=relu	(4096)	102,764,544
fc2 (Dense)	units=4096, activation=relu	(4096)	16,781,312
predictions (Dense)	units=1000, activation=softmax	(1000)	4,097,000
Model	Parameters: total=143,667,240, trainable=143,667,240		

2.2.4 Inception Modules and GoogLeNet

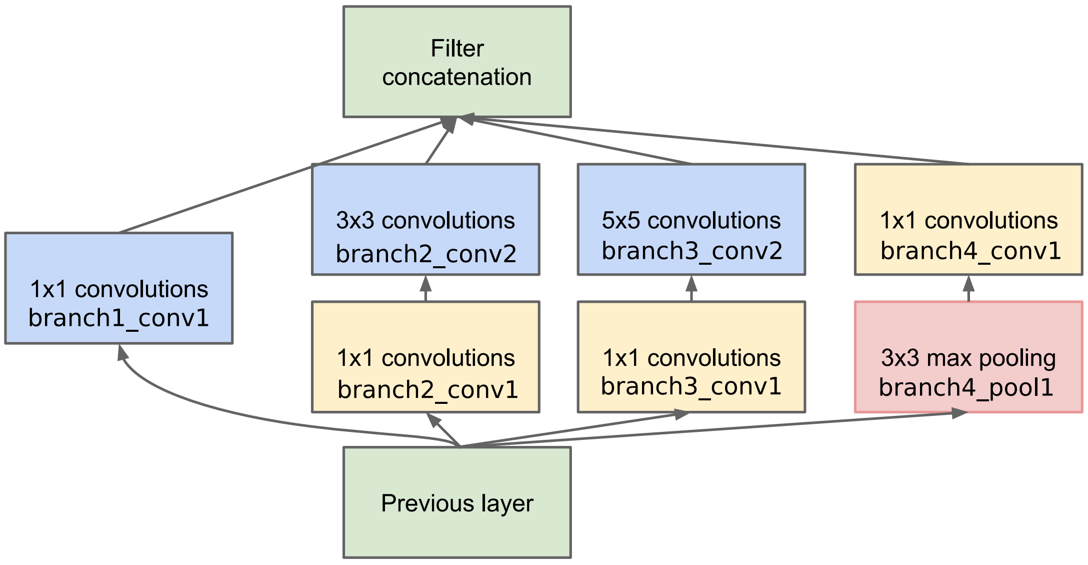

Filter concatenation
1x1 convolutions branch1_conv1
3x3 convolutions branch2_conv2
5x5 convolutions branch3_conv2
1x1 convolutions branch4_conv1
1x1 convolutions branch2_conv1
1x1 convolutions branch3_conv1
3x3 max pooling branch4_pool1
Previous layer

Figure 2.4 – The inception module proposed by (Szegedy et al., 2015). Figure from (Szegedy et al., 2015).

Szegedy et al. (2015) proposed the inception modules to increase the depth of the networks even further without significantly increasing the number of parameters and the computational cost of the networks. The inception modules use parallel convolutions of different filter sizes at each layer to introduce mutli-scale processing and sparse computations. They also use 1×1 convolutions (Lin, Chen, and Yan, 2013) to reduce the number of parameters in the inception modules. The diagram of the inception modules is shown in figure 2.4. Details of one configuration of the inception module, inception (3a) in GoogLeNet in (Szegedy et al., 2015), are shown in table A.1 on page 129.

The GoogLeNet architecture[3] has 22 weight layers using the inception modules. Its details are shown in table 2.6. This network performs significantly better on image classification compared to AlexNet (8 layers, 60 million parameters) and VGG-19 (19 layers, 144 million parameters) while having much less parameters, 7 million.

2.2.5 Batch normalization

Introduced by Ioffe and Szegedy (2015), batch normalization accelerates the training of deep networks by addressing the **internal covariate shift** (Ioffe and Szegedy, 2015) which is the distribution change of features at each layer that happens during training of deep neural networks. Internal covariate shift can cause **vanishing or exploding gradients** which can either slow down the training or make the training impossible altogether (Ioffe and Szegedy,

[3]This architecture is referred as *Inception v1* in some publications.

Table 2.6 – Architecture details of GoogLeNet (also known as inception v1) presented in (Szegedy et al., 2015). The input to the architecture are color 224 × 224 pixel images. *LRN* refers to local response normalization presented in (Krizhevsky, Sutskever, and G. E. Hinton, 2012). The *InceptionModule* is presented in figure 2.4 and a configuration of it, inception (3a), is shown in table A.1 on page 129. The number of filters for each convolution in the inception modules are shown in the details column.

Layer (type)	Details	Output Shape	Number of Parameters
conv1/7x7_s2 (Conv2D)	filters=64, kernel_size=(7, 7), strides=(2, 2), padding=same, activation=relu	(112, 112, 64)	9,472
pool1/3x3_s2 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=same	(56, 56, 64)	0
pool1/norm1 (LRN)	depth_radius=5, alpha=0.0001, beta=0.75	(56, 56, 64)	0
conv2/3x3_reduce (Conv2D)	filters=64, kernel_size=(1, 1), strides=(1, 1), padding=same, activation=relu	(56, 56, 64)	4,160
conv2/3x3 (Conv2D)	filters=192, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(56, 56, 192)	110,784
conv2/norm2 (LRN)	depth_radius=5, alpha=0.0001, beta=0.75	(56, 56, 192)	0
pool2/3x3_s2 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=same	(28, 28, 192)	0
inception (3a) (InceptionModule)	b1_c1=64, b2_c1=96, b2_c2=128, b3_c1=16, b3_c2=32, b4_c1=32	(28, 28, 256)	163,696
inception (3b) (InceptionModule)	b1_c1=128, b2_c1=128, b2_c2=192, b3_c1=32, b3_c2=96, b4_c1=64	(28, 28, 480)	388,736
pool3/3x3_s2 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=same	(14, 14, 480)	0
inception (4a) (InceptionModule)	b1_c1=192, b2_c1=96, b2_c2=208, b3_c1=16, b3_c2=48, b4_c1=64	(14, 14, 512)	376,176
inception (4b) (InceptionModule)	b1_c1=160, b2_c1=112, b2_c2=224, b3_c1=24, b3_c2=64, b4_c1=64	(14, 14, 512)	449,160
inception (4c) (InceptionModule)	b1_c1=128, b2_c1=128, b2_c2=256, b3_c1=24, b3_c2=64, b4_c1=64	(14, 14, 512)	510,104
inception (4d) (InceptionModule)	b1_c1=112, b2_c1=144, b2_c2=288, b3_c1=32, b3_c2=64, b4_c1=64	(14, 14, 528)	605,376
inception (4e) (InceptionModule)	b1_c1=256, b2_c1=160, b2_c2=320, b3_c1=32, b3_c2=128, b4_c1=128	(14, 14, 832)	868,352
pool4/3x3_s2 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=same	(7, 7, 832)	0
inception (5a) (InceptionModule)	b1_c1=256, b2_c1=160, b2_c2=320, b3_c1=32, b3_c2=128, b4_c1=128	(7, 7, 832)	1,043,456
inception (5b) (InceptionModule)	b1_c1=384, b2_c1=192, b2_c2=384, b3_c1=48, b3_c2=128, b4_c1=128	(7, 7, 1024)	1,444,080
pool5 (GlobalAveragePooling2D)		(1024)	0
dropout (Dropout)	drop_rate=0.4	(1024)	0
output (Dense)	units=1000, activation=softmax	(1000)	1,025,000
Model	Parameters: total=6,998,552, trainable=6,998,552		

2015; Nair and G. E. Hinton, 2010; Glorot and Y. Bengio, 2010). During the training steps of the network, batch normalization normalizes the features of a layer by subtracting the features by their mean and dividing them by their standard deviation. The normalization is done for each dimension of features independently and the mean and variance of features is estimated over a *mini-batch*. This normalization reduces the effect of internal covariate shift. However, normalizing the features of a layer to have a mean of zero and standard deviation of one may change what the features can represent. To avoid this, batch normalization also introduces two trainable parameters: β and γ which make sure that the total transformation done by batch normalization can represent the identity function. Moreover, a moving average of the mean and standard deviation of each feature is calculated during training which estimates the mean and standard deviation of the whole dataset. These values are used to normalize the features at test time. The formulation of batch normalization is given below.

If a layer represents the following function:

$$\boldsymbol{z} = \mathrm{g}(\mathbf{W}\boldsymbol{u} + \mathbf{b}) \tag{2.8}$$

where $\boldsymbol{u}$ is the input, $\boldsymbol{z}$ is the output, $\mathbf{W}$ and $\mathbf{b}$ are the trainable parameters the layer, and g is a nonlinearity function such as ReLU. Then, batch normalization changes the layer to[4]:

$$\boldsymbol{z} = \mathrm{g}(BN(\mathbf{W}\boldsymbol{u})) \tag{2.9}$$

where BN is the batch normalization transformation. During training and given a mini-batch

[4]Note that **b** is not needed as it will be subsumed by β.

of size m, $\mathbb{B} = \{x_{1...m}\}$, the BN transformation of x_i, a feature in a mini-batch, is given as:

$$BN(x_i) = \gamma \widehat{x_i} + \beta \tag{2.10}$$

$$\widehat{x_i} = \frac{x_i - \mu_{\mathbb{B}}}{\sqrt{\sigma^2_{\mathbb{B}} + \epsilon}} \tag{2.11}$$

$$\sigma^2_{\mathbb{B}} = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathbb{B}})^2 \tag{2.12}$$

$$\mu_{\mathbb{B}} = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{2.13}$$

where ϵ is a small number which is used for numerical stability. $BN(x_i)$ is calculated for each dimension of features independently. This transformation is directly applicable to fully-connected layers, however, a small change is needed for convolutional layers. During the training of convolutional layers, feature maps have the size of $m \times p \times q \times k$ in a mini-batch where m is the batch size, p and q are the spatial dimensions, and k is the dimension of the features. Then, in equations (2.12) and (2.13), m is replaced by $m.p.q$ which means that the mean and standard deviation of each feature map is calculated over the mini-batch and the spatial dimensions as if they form a larger mini-batch. At test time, the mean and variance in equations (2.12) and (2.13) is replaced by the mean and standard deviation of the whole training dataset which is estimated using moving averages during training.

Adding the batch normalization transformation to networks brings several advantages (Ioffe and Szegedy, 2015):

- The training will be less sensitive to hyper-parameters and learning-rate can be increased to speed-up the training. This is possible due to alleviation of internal covariate shift.

- During training, the value of normalized features of each input sample is not deterministic because it depends on other samples in the mini-batch. This is believed to regularize the network and reduces the chances of overfitting.

2.2.6 Residual Connections and ResNets

K. He et al. (2016) show that increasing the depth of a CNN, improves the performance of the CNN initially. However, as the depth increases to 30 layers or more, the performance of the network degrades. This performance degradation occurs even on the training data so overfitting can be ruled out. This problem is referred as the **degradation problem**. The deep residual learning framework proposed by K. He et al. (2016) allows us to increase the depth of the networks even further, hundreds of layers deep, while still being able to avoid the degradation problem. To overcome the degradation problem, they introduce residual connections where the input of a layer is added to the output of a deeper layer. For example, if

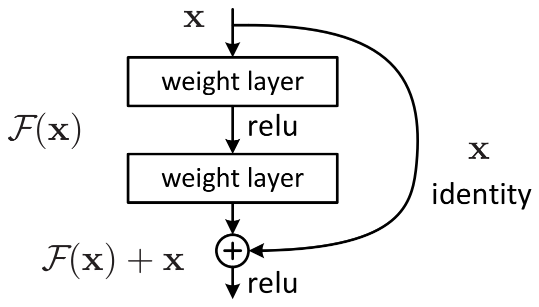

x
weight layer
$\mathcal{F}(\mathrm{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathrm{x}) + \mathrm{x}$
relu

Figure 2.5 – A residual connection. The input of function F is added to its output to allow easier learning of deep networks. Figure from K. He et al. (2016).

a layer (or multiple layers) represent a function $\mathrm{F}(\boldsymbol{x})$[5], to make it residual, we replace $\mathrm{F}(\boldsymbol{x})$ with $\mathrm{H}(\boldsymbol{x})$ where:

$$\mathrm{H}(\boldsymbol{x}) = \mathrm{F}(\boldsymbol{x}) + \boldsymbol{x} \tag{2.14}$$

A diagram of a residual connection is shown in figure 2.5. The residual learning framework allowed the authors to develop ResNet-152 that with 152 weight layers is significantly deeper compared to VGG or Inception networks. The authors, using ResNet-152 with 60 million parameters, won the image classification, detection, and localization competitions in ILSVRC 2015 (K. He et al., 2016).

2.2.7 Scaled Residuals and Inception Resnets

Inception networks (Szegedy et al., 2015; Ioffe and Szegedy, 2015; Szegedy, Vanhoucke, et al., 2016) perform well on the image classification task with less parameters and computations compared to traditional CNNs such as VGG networks. On the other hand, residual networks (K. He et al., 2016) are much deeper architectures consisting of traditional convolutional layers which perform as well as the inception networks. This has encouraged Szegedy, Ioffe, et al. (2016) to investigate the effect of adding residual connections to Inception networks. They show that residual counterparts of inception networks train significantly faster compared to pure inception networks. Moreover, they also introduce the concept of *scaling* in residual connections to stabilize the training of residual networks when the networks become too wide (*e.g.,* more than 1000 filters in convolutional layers). A scaled residual connection, using the same notation as residual connections, may be seen as:

$$\mathrm{H}(\boldsymbol{x}) = \alpha\mathrm{F}(\boldsymbol{x}) + \boldsymbol{x} \tag{2.15}$$

where $\boldsymbol{x}$ is the input, F is the function that represents a layer or multiple layers, H is the output, and α is the scale factor. Szegedy, Ioffe, et al. (2016) used values between 0.1 and 0.3 for α.

[5]Note that $\mathrm{F}(\boldsymbol{x})$ must be a non-linear function otherwise its residual variant can be estimated using another linear function.

Moreover, Szegedy, Ioffe, et al. (2016) introduce the InceptionResNetV2 architecture which has improvements in all aspects compared to former inception networks (Szegedy et al., 2015; Ioffe and Szegedy, 2015; Szegedy, Vanhoucke, et al., 2016) and also adds residual connections. This architecture is detailed in table 2.7. Its blocks are detailed in tables A.2 to A.8 on pages 129 to 131. Compared to the original inception module shown in table A.1 and figure 2.4, the new InceptionResNetV2 architecture introduces several new inception blocks where they have different number of branches, the depth of some branches has increased, and also the kernel sizes in the convolutional layers have changed. Moreover, the depth of the InceptionResNetV2 is increased significantly compared to the GoogLeNet (Inception v1) architecture shown in table 2.6.

2.2.8 Dense Connections and DenseNets

Inspired by residual connections (K. He et al., 2016), G. Huang et al. (2017) proposed dense connections. In dense connections, output of all preceding layers is used as input for the next layer. Assume that x_0 is the input to an L layer *densely* connected network, each layer represents a function, H_l, and x_l is the output of the lth layer. Then, the output of each layer is calculated as:

$$x_l = \mathrm{H}_l([x_0, x_1, ..., x_{l-1}]) \tag{2.16}$$

where the [...] operator is the channel-wise concatenation of feature maps. The diagram of dense connections is shown in figure 2.6.

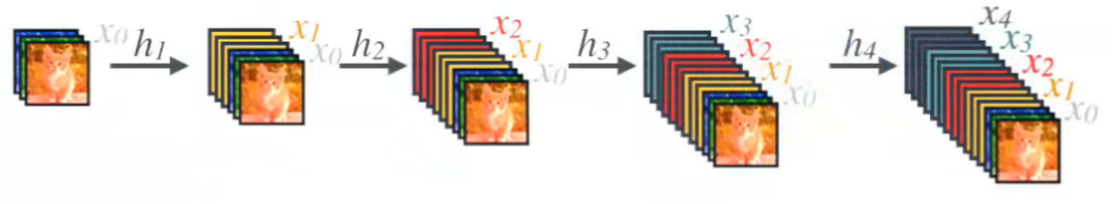


Figure 2.6 – A densely connected network. Each layer takes as input the output of *all* preceding layers. x_i is the output of ith layer and h_i is the non-linear function (including the learned weights) that layer i applies on its input. Output of h_i, x_i, is concatenated (channel-wise) with the output of all preceding layers and is used as input to the next layer. Figure from https://youtu.be/-W6y8xnd--U?t=157 by (G. Huang et al., 2017).

G. Huang et al. (2017) also chose a standard function for H_l, referred as a *ConvBlock*, which is composed of batch normalization, ReLU, and a 3×3 convolution. A ConvBlock can optionally have a bottleneck layer which uses 1×1 convolutions to reduce the number of filters before applying the 3×3 convolution. Several ConvBlocks can be densely connected to each other to construct a *DenseBlock*. Because DenseBlocks keep the spatial dimensions of feature maps constant, a *TransitionBlock* is also introduced to be placed between DenseBlocks to reduce the spatial dimensions of feature maps. A TransitionBlock is made of batch normalization, ReLU, 1×1 convolution, and average pooling. Using these building blocks, the authors proposed DenseNet-161, with 29 million parameters, which achieves the same accuracy on ImageNet image classification as ResNet-152 but with half the number of parameters. The architecture details of DenseNet-161 is shown in table 2.8. The details of its first DenseBlock, its first ConvBlock, and its first TransitionBlock are shown in tables A.9, A.10 and A.11, respectively on page 132. This architecture details are useful further in the thesis when one of the face PAD baselines is explained. G. Huang et al. (2017) empirically show that DenseNets are less prone to overfitting compared to ResNets, possibly due to feature reuse properties of the network.

Table 2.7 – Architecture details of InceptionResNetV2 presented in (Szegedy, Ioffe, et al., 2016). This architecture is slightly different from what was published in (Szegedy, Ioffe, et al., 2016) but is treated as the official version. The input to the architecture are color 299 × 299 pixel images. Details of *conv2d_bn (Conv2D_BN)*, *inception_a (InceptionA)*, *block35_1 (InceptionResnetBlock)*, *block17_1 (InceptionResnetBlock)*, *block8_1 (InceptionResnetBlock)*, *inception_a (InceptionA)*, and *reduction_b (ReductionB)* are shown in tables A.2, A.3, A.4, A.5, A.6, A.7 and A.8, respectively on pages 129 to 131.

Layer (type)	Details	Output Shape	Number of Parameters
input_2 (InputLayer)		[(299, 299, 3)]	0
conv2d_bn (Conv2D_BN)	filters=32, kernel_size=3, strides=2	(149, 149, 32)	960
conv2d_bn_1 (Conv2D_BN)	filters=32, kernel_size=3, strides=1	(147, 147, 32)	9,312
conv2d_bn_2 (Conv2D_BN)	filters=64, kernel_size=3, strides=1	(147, 147, 64)	18,624
max_pooling2d (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(73, 73, 64)	0
conv2d_bn_3 (Conv2D_BN)	filters=80, kernel_size=1, strides=1	(73, 73, 80)	5,360
conv2d_bn_4 (Conv2D_BN)	filters=192, kernel_size=3, strides=1	(71, 71, 192)	138,816
max_pooling2d_1 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(35, 35, 192)	0
inception_a (InceptionA)	pool_filters=64	(35, 35, 320)	268,848
block35_1 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_2 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_3 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_4 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_5 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_6 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_7 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_8 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_9 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
block35_10 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=1	(35, 35, 320)	123,408
reduction_a (ReductionA)	k=256, kl=256, km=384, n=384	(17, 17, 1088)	2,666,240
block17_1 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_2 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_3 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_4 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_5 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_6 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_7 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_8 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_9 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_10 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_11 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_12 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_13 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_14 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_15 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_16 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_17 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_18 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_19 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
block17_20 (InceptionResnetBlock)	block_type=block17, scale=0.1, n=1	(17, 17, 1088)	1,127,456
reduction_b (ReductionB)	k=256, kl=288, km=320, n=256, no=384, p=256, pq=288	(8, 8, 2080)	3,883,008
block8_1 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_2 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_3 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_4 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_5 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_6 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_7 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_8 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_9 (InceptionResnetBlock)	block_type=block8, scale=0.2, n=1	(8, 8, 2080)	2,036,288
block8_10 (InceptionResnetBlock)	block_type=block8, scale=1.0, n=1	(8, 8, 2080)	2,036,288
conv_7b (Conv2D_BN)	filters=1536, kernel_size=1, strides=1	(8, 8, 1536)	3,199,488
avg_pool (GlobalAveragePooling2D)		(1536)	0
predictions (Dense)	units=1000, activation=softmax	(1000)	1,537,000
Model	Parameters: total=55,873,736, trainable=55,813,192		

Table 2.8 – Architecture details of DenseNet-161 presented in (G. Huang et al., 2017). Details of *dense_block_1* and *transition_block_1* are shown in tables A.9 and A.11, respectively on page 132. In DenseBlocks, *layers* is the number of ConvBlocks, *growth rate* is the number of filters for each 3 convolution inside a ConvBlock. See table A.10 on page 132 for details of *bottleneck* and *dropout rate* parameters. In TransitionBlocks, *filters* is the number of filters in $1x1$ convolutions. The input to the architecture are color 224 × 224 pixel images.

Layer (type)	Details	Output Shape	Number of Parameters
conv0_pad (ZeroPadding2D)	padding=((3, 3), (3, 3))	(230, 230, 3)	0
conv0 (Conv2D)	filters=96, kernel_size=(7, 7), strides=(2, 2), padding=valid	(112, 112, 96)	14,112
norm0 (BatchNormalization)		(112, 112, 96)	384
relu0 (Activation)	activation=relu	(112, 112, 96)	0
pool0_pad (ZeroPadding2D)	padding=((1, 1), (1, 1))	(114, 114, 96)	0
pool0 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(56, 56, 96)	0
dense_block_1 (DenseBlock)	layers=6, growth_rate=48, bottleneck=True, dropout_rate=0	(56, 56, 384)	756,288
transition_block_1 (TransitionBlock)	filters=192	(28, 28, 192)	75,264
dense_block_2 (DenseBlock)	layers=12, growth_rate=48, bottleneck=True, dropout_rate=0	(28, 28, 768)	2,077,056
transition_block_2 (TransitionBlock)	filters=384	(14, 14, 384)	297,984
dense_block_3 (DenseBlock)	layers=36, growth_rate=48, bottleneck=True, dropout_rate=0	(14, 14, 2112)	11,650,176
transition_block_3 (TransitionBlock)	filters=1056	(7, 7, 1056)	2,238,720
dense_block_4 (DenseBlock)	layers=24, growth_rate=48, bottleneck=True, dropout_rate=0	(7, 7, 2208)	9,573,120
norm5 (BatchNormalization)		(7, 7, 2208)	8,832
relu5 (Activation)	activation=relu	(7, 7, 2208)	0
last_pool (GlobalAveragePooling2D)		(2208)	0
classifier (Dense)	units=1000	(1000)	2,209,000
Model	Parameters: total=28,900,936, trainable=28,681,000		

2.2.9 Autoencoders

In chapter 6, CNN-based **autoencoders** are used. Below, an introduction to autoencoders is given for reference. An autoencoder is an artificial neural network used for mainly dimensionality reduction as shown in figure 2.7. During training, the input is fed to a feed-forward network and the expected output is also the same input. In most common autoencoders, the number of nodes in an autoencoder decrease from input to the middle and increase again till the output (Y. Bengio, 2009). At the layer where the number of nodes are smallest, the network learns a compact **representation** of data (also called **hidden codes** or **latent variable**; see **z** in figure 2.7) by learning to construct the same input at the last layer. This representation of data is usually used as a set features in a back-end classifier. Normally, autoencoders are made of Dense layers. However, when the inputs are spatially correlated, such as images, CNN layers are used as well (Y. Bengio, Courville, and Vincent, 2013). Below, the details of Info-VAEs, a family of autoencoders used in chapter 6, are given.

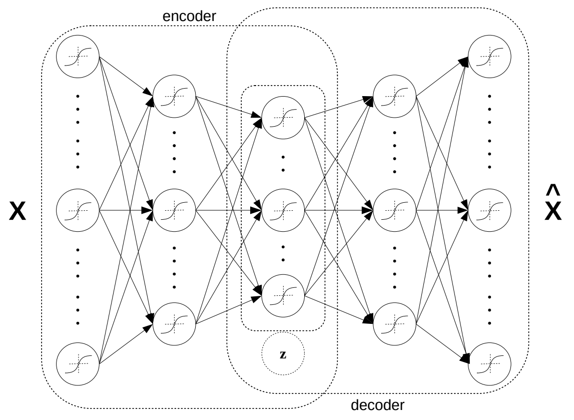


Figure 2.7 – An autoencoder. The input is **x** and the output is the reconstructed version of **x**, $\hat{\mathbf{x}}$. **z** is the representation (also called *hidden codes* or *latent variable*) of the input in a much lower dimension. The first part encodes the data and the second part decodes the data.

Information maximizing variational autoencoders (Info-VAE) are a family of autoencoders that are able to learn *meaningful and disentangled* representations of samples where each dimension in the learned representation can represent one underlying factor of the data (S. Zhao, Song, and Ermon, 2017). Info-VAEs learn these representations by matching the aggregated posterior of the latent variable to an arbitrary prior distribution. **Adversarial AutoEncoders (AAE)** by (Makhzani et al., 2015) are one implementation of Info-VAEs. They match the aggregated posterior distribution of the latent variable to the prior distribution through the use of the *generative adversarial network (GAN)* framework (Goodfellow et al., 2014). Figure 2.8 outlines the training process of an AAE. During training, the autoencoder is trained with two objectives: the traditional reconstruction loss and an adversarial loss similar to GANs. The discriminator is a neural network that learns to distinguish between real samples (samples from p(**z**)) from fake samples (samples from q(**z**|**x**)). The encoder part is trained with two objectives: outputting meaningful hidden codes that the decoder can use to reconstruct the original input and outputting hidden codes that are indistinguishable from samples of p(**z**).

As a result the decoder can output meaningful samples from any data that is sampled from the prior distribution.

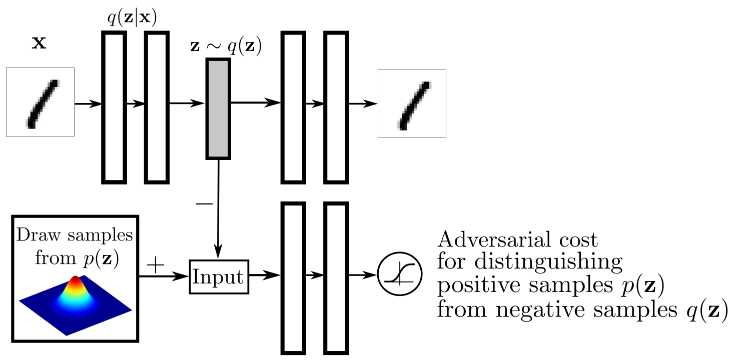

$\mathbf{x}$
$q(\mathbf{z}|\mathbf{x})$
$\mathbf{z} \sim q(\mathbf{z})$
$-$
Draw samples from $p(\mathbf{z})$
$+$
Input
Adversarial cost for distinguishing positive samples $p(\mathbf{z})$ from negative samples $q(\mathbf{z})$

Figure 2.8 – An adversarial autoencoder (AAE). AAEs impose a prior distribution, p($\mathbf{z}$), on hidden codes using a discriminator and an adversarial loss. The discriminator learns to distinguish real samples (samples from p($\mathbf{z}$)) from fake samples (samples from q($\mathbf{z}$|$\mathbf{x}$)) and the encoder learns to output hidden codes that are indistinguishable from real samples. Figure from (Makhzani et al., 2015).

2.3 Face Recognition (FR)

Progress of face recognition (FR) technology, from strongly constrained models to fully unconstrained environments, has been enabled by the adoption of successively complex learning paradigms. The first generation of large scale appearance based FR systems, such as Eigenfaces (Turk and Pentland, 1991) and Fisherfaces (Wiskott et al., 1997), attempted to model the face-variability in a simple linear sub-space. Subsequently, methods such as joint factor analysis (JFA), inter-session variability (ISV) modeling (Wallace et al., 2011) and probabilistic linear discriminant analysis (PLDA) (Prince and Elder, 2007; El Shafey et al., 2013) were developed to better model variability in face-images. Deep-learning based methods, notably convolutional neural networks (CNN) (Taigman et al., 2014; Parkhi, Vedaldi, and Zisserman, 2015; Schroff, Kalenichenko, and Philbin, 2015; Y. Sun et al., 2015), have become very popular in recent years, due to their near-perfect recognition accuracy on unconstrained datasets such as 'labeled faces in the wild' (LFW) (G. B. Huang et al., 2007).

A FR system may be used in two kinds of applications: *face-identification* or *face-verification*. Face-identification is a one-to-many problem, where the face-image to be identified is tested against all previously enrolled identities, to see if it matches any of the known identities. In face-verification systems, a claimed-identity is provided along with the input face-image, and the problem is simply to verify that the input image corresponds to the claimed-identity. In this work, the terms *recognition* and *verification* have been used interchangeably, and refer to the use of a FR system in verification mode.

A typical FR system functions in three phases: training, enrollment, and probing. In the training phase a background model, assumed to broadly represent the space of face-images, is constructed using training data. In the enrollment phase, the FR system generates templates for the given enrollment samples, which are then stored in the gallery. In the probing (operational) phase, the FR system is presented with a probe-image and a claimed identity. A template is created for the given probe-sample, and is compared with the set of enrolled templates associated with the claimed identity. The result of the comparison is a score, which is then thresholded to produce a decision (accept or reject).

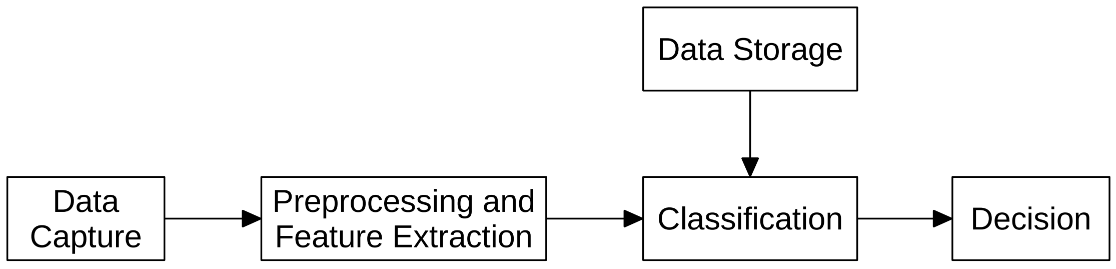


Figure 2.9 – Diagram of a biometric system.

A biometric system (here, an FR system) is usually made of several parts as shown in figure 2.9 (ISO/IEC DIS 30107-1, 2016). A *data capture* module is responsible for the capturing the data, images of faces in FR. A *feature extractor* module processes the captured data and extracts features (also called **templates**) from the processed data. A *data storage* module is used to store previously enrolled templates. When a probe-sample is presented to the system, its template is compared with the enrolled templates using a *classification* module. Finally, a *decision* module is used to output a decision based on the results of the classification module.

To mitigate the influence of variations on the actual face-recognition process, the raw input image is usually preprocessed, to extract sub-images representing individual faces. Geometric and color transforms may also be applied to the extracted face-images, depending on the requirements of the specific FR method. The result of the pre-processing stage is a *normalized* face image, of predefined size and scale, that may be processed by a FR system. Before describing the different FR methods, we explain the pre-processing steps applied to normalize the input face images.

2.3.1 Face Image Normalization

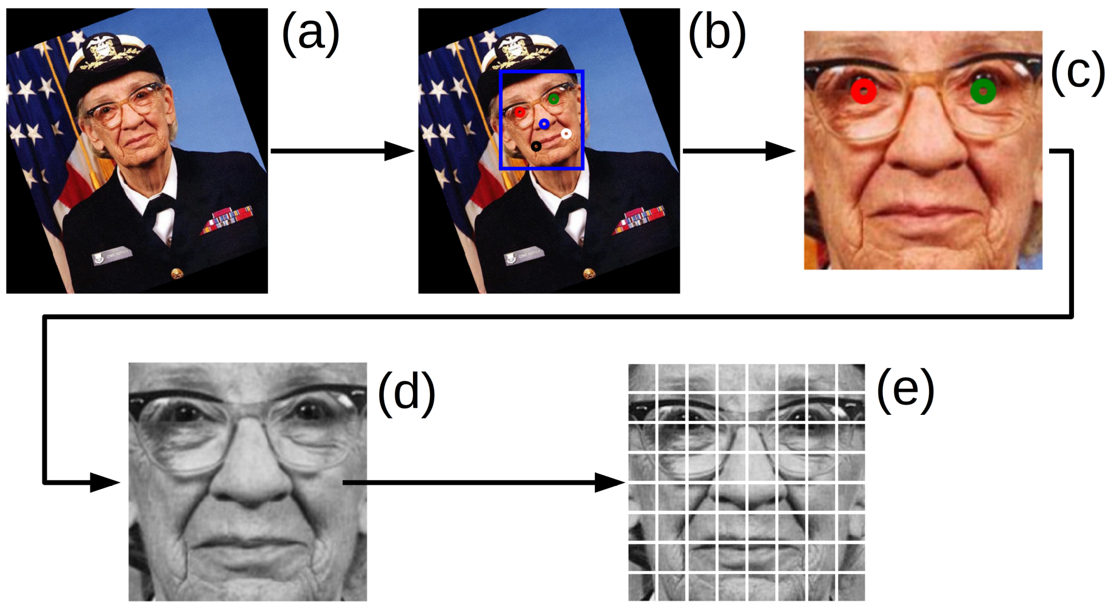


Figure 2.10 – A face image normalization process. The original image is shown in (a). The boundaries of the detected face is shown as a blue box in (b). Located face landmarks are shown as circles in (b). The result of cropping and geometrically normalizing the face image is shown in (c). Usually faces are normalized so that the eye centers fall in the expected locations (c). The face image may be converted to a gray-scale image (d). Patches of the face image may be extracted from the face image (e). The individual patches may be used as input to a system instead of the full face image. Note that patches may be extracted from the color face image as well. Note that depending on the system, only a few of these steps may be applied.

Different FR (and PAD) systems expect the face input image in different formats. Usually, the original face image needs to be processed and normalized before it can be used as input to an FR system. This process typically involves the following steps:

- **face detection**: where the location of the face in the image is determined.
- **landmark localization**: where the location of some specific landmarks, such as the location of eye centers, of the face is determined.
- **face cropping and alignment**: where the location of landmarks are used to crop and geometrically normalize the face image. Usually, the face images are geometrically transformed so that the eye centers fall on predefined locations.
- **further processing**: the face image may be further processed. It may be converted to a gray scale image or small **patches** may be extracted from the face image.

The face normalization process is demonstrated in figure 2.10. The original input image is shown in figure 2.10.a. Results of face detection and landmark localization are shown in figure 2.10.b. Results of face cropping is shown in figure 2.10.c. Results of further possible transformations are shown in figure 2.10.d and figure 2.10.e.

In our experiments, the normalized face-region is extracted using annotations identifying the center of each eye. Imposing the constraints that the straight-line joining the two eye-centres should be horizontal, and should have a predefined length, an affine transform can be used to extract a normalized face-image of fixed size from the given input image. Some face-biometrics test datasets include annotations for the eye locations. For datasets where this information is not explicitly provided, a CNN-based algorithm is used for face detection and landmark localization (K. Zhang et al., 2016).

2.3.2 CNN-Based FR Systems

CNNs (LeCun et al., 1998; Goodfellow, Y. Bengio, and Courville, 2016) (see section 2.1 on page 9), a class of deep neural networks (DNN), have been shown to be extremely accurate in FR tasks. For FR applications, CNNs are usually trained for face identification, using face-images as input, and the set of identities to be recognized as output. The last few layers of the network, including the output layer, are typically fully-connected (*fc* for short) layers (also called dense layers). These layers may be seen as the classifier-stage of the network, whereas the preceding (convolutional and pooling) layers may be considered to constitute the feature-extraction stage. Although CNNs are typically trained end-to-end for classification or regression tasks, they are often also used as feature-extraction tools. The terms *representation* and *embedding* are used interchangeably, to denote the outputs of the various layers of a deep network. Representations generated by a specific layer of a pre-trained DNN may be used as templates (feature-vectors) representing the corresponding input images. These templates may be subsequently be used to train a classifier, or to compare the respective input images using appropriate similarity measures.

(Taigman et al., 2014) have proposed DeepFace, a 9-layer CNN for FR. The input faces are first aligned to have an upright position using 3D modeling and piecewise affine transforms, before being fed to the network. This network achieves a recognition accuracy of 97.25% on the LFW dataset (G. B. Huang et al., 2007). (Schroff, Kalenichenko, and Philbin, 2015) report an accuracy of 99.63% on the LFW dataset using FaceNet, a CNN with 7.5 million parameters, trained using a novel *triplet* loss function. Other CNN architectures showing similar FR accuracy on the LFW dataset include the DeepID series by (Y. Sun et al., 2015).

In this work we describe three CNN-FR methods: the VGG-Face (Parkhi, Vedaldi, and Zisserman, 2015), LightCNN (X. Wu et al., 2015), and FaceNet (Sandberg, 2017; Schroff, Kalenichenko, and Philbin, 2015). The VGG-Face network has been included in our study because it is a very well known and widely referenced CNN for FR applications. The other two network models have been included because they are newer than the VGG-Face network, and have both shown a FR performance even better than that of the VGG-Face CNN. Another important reason why these three specific CNNs have been studied in this work is that the respective creators of these networks have made publicly available pre-trained models that can be directly used in our experiments. The following sections provide brief summaries of the selected CNNs, and describe how they have been used in our experiments.

2.3.2.1 VGG-Face CNN

The VGG-Face network model is made publicly available by the Visual Geometry Group[6] at Oxford University. Involving almost 135 million trainable parameters, this network has been shown to achieve a FR accuracy of 98.95% on the LFW unrestricted setting (G. B. Huang et al., 2007). VGG-Face is a CNN consisting of 16 hidden layers (see Table 3 in (Parkhi, Vedaldi, and Zisserman, 2015)). The initial 13 hidden layers are convolution and pooling layers, and the last three layers are fully-connected ('fc6', 'fc7', and 'fc8', following the nomenclature used by (Parkhi, Vedaldi, and Zisserman, 2015)). The input to this network is an appropriately cropped color face-image of pre-specified dimensions.

We use the representation produced by the 'fc7' layer of the VGG-Face CNN as a template for the input image. When enrolling a client, the template produced by the VGG-Face network for each enrollment-sample is recorded. For verification, the network is used to generate a template for the probe face-image, which is then compared to the enrolled templates of the claimed identity using the Cosine-similarity measure given by equation (2.17) (where $||\boldsymbol{a}||$ represents the L^2-norm of vector $\boldsymbol{a}$).

$$Cosine\_Similarity(\boldsymbol{a},\boldsymbol{b}) = \frac{\boldsymbol{a}\cdot\boldsymbol{b}}{||\boldsymbol{a}||\cdot||\boldsymbol{b}||} \tag{2.17}$$

The score assigned to the probe is the average Cosine-similarity of the probe-template to all the enrollment-templates of the claimed identity. If the score is larger than a predetermined threshold, the probe is accepted as a match for the claimed identity.

[6]Website: www.robots.ox.ac.uk/~vgg/software/vgg_face

2.3.2.2 LightCNN

(X. Wu et al., 2015) have proposed a new CNN, called LightCNN, for FR. Their goals in designing this network were to have significantly fewer trainable parameters compared to other state-of-the-art FR-CNNs, as well as to be able to handle noisy labels that are inevitable in datasets mined automatically from the web. Compared to the VGG-Face network, the number of parameters in the LightCNN model is smaller by a factor of 10 (~ 12 million parameters). This is achieved mainly through the use of a newly introduced Max-Feature-Map (MFM) activation (X. Wu et al., 2015), which is a non-linear extension of the maxout activation operation. Although the MFM operator is more expensive to compute than the ReLU unit that it replaces, the large overall reduction of number of units per layer, made possible by the use of the new operator, still leads to smaller computation time for the forward-pass of the LightCNN network (by a factor of 5, relative to the VGG-Face CNN (X. Wu et al., 2015)).

In our experiments we use as templates the 256-D representation produced by the 'eltwise_fc1' layer of LightCNN. Note that this template is much smaller than the 4096-D vector produced by VGG-Face. Despite these relative efficiencies, the LightCNN network achieves a FR accuracy of 99.33% on the LFW dataset (in unrestricted setting) – outperforming the VGG-Face network by a small margin. As with the VGG-Face network, the Cosine measure (equation (2.17)) is used to compare an input probe template to the relevant enrollment templates.

2.3.2.3 FaceNet CNN

Very recently, David Sandberg has made publicly available his implementation as well as trained models for a new FR-CNN named *FaceNet*(Sandberg, 2017). This is the closest open-source implementation of the FaceNet CNN proposed by (Schroff, Kalenichenko, and Philbin, 2015), for which neither a pre-trained model nor the training-set is publicly available. Sandberg's FaceNet implements an Inception-ResNetV1 DNN architecture (Szegedy, Ioffe, et al., 2016). Several FaceNet models have been published (Sandberg, 2017). In our tests, we have used the 20170512-110547 model, trained on the MS-Celeb-1M dataset (Guo et al., 2016). Using this model, FaceNet achieves a FR performance of 99.2% on the LFW dataset (Sandberg, 2017), which is comparable to the performance of LightCNN. Note that the 128-D representation produced by this network (at the 'embeddings:0' layer) is half the size of that produced by LightCNN. We use this representation to construct enrollment and probe templates, which are compared to each other using the Cosine measure (equation (2.17)).

2.3.3 GMM-based FR using Inter-Session Variability Modeling

Inter-session variability modeling (ISV) is an extension of the Gaussian Mixture Models (GMM) based method for face verification. We have used the ISV modeling approach proposed by (Wallace et al., 2011). Among the FR methods included in this study, this is the only method that adopts a *parts based* approach. The input normalized face image is first decomposed into a set of square sub-images of size (12 × 12), with an overlap of 11 pixels in each direction. Let N represent the total number of sub-images extracted from the input face-image. For each sub-image, a predetermined number, D, of low-frequency DCT (discrete cosine transform) coefficients is computed. Thus, a set of N D-dim arrays of DCT coefficients is extracted for each input face-image. The DCT coefficients are normalized to zero-mean and unit standard-deviation in each of the D dimensions. The resulting set of N normalized D-dim DCT arrays is used to represent the input face-image.

To use a GMM for FR, first, a universal background model (UBM) is constructed from the set of training-images (each represented by N D-dim DCT arrays). The UBM is a GMM, that is, a weighted sum of K Gaussians, where each Gaussian is represented by a D-dim mean-vector, and a $(D \times D)$-dim covariance-matrix. The parameters of the Gaussians components of the UBM, as well as their relative weights, are learned from the training data. The UBM describes the probability distribution of face sub-images in the D-dim DCT-coefficient space.

A *supervector*, $\mathbf{m}$, is then constructed by concatenating K the mean-vectors of all the components of the GMM. To enroll a client i, a supervector $\mathbf{s}_i$ is derived as follows:

$$\mathbf{s}_i = \mathbf{m} + \mathbf{d}_i \ , \tag{2.18}$$

where $\mathbf{d}_i$ is an offset-vector specific to client i, computed using maximum *a posteriori* (MAP) adaptation from the UBM (Reynolds, Quatieri, and Dunn, 2000).

The supervector $\mathbf{s}_i$ is assumed to be client-specific. In other words, different enrollment images of the same client, i, should, in theory, produce very similar supervectors. For a given client, however, enrollment set typically contains images captured in different sessions, with varying pose, illumination, and facial expressions. This within-class variability for client i is also reflected in $\mathbf{s}_i$, and can result in diminished recognition accuracy of the GMM-based FR method (Wallace et al., 2011).

ISV-modeling has been developed to enhance the GMM based approach by explicitly modeling and suppressing the within-class variability of each enrolled client. We assume that each enrollment image, collected in a separate session, results in a unique supervector. For enrollment image (i, j) (the j^{th} enrollment image of client i), let

$$\boldsymbol{\mu}_{i,j} = \mathbf{m} + \mathbf{u}_{i,j} + \mathbf{d}_i \ , \tag{2.19}$$

where $\boldsymbol{\mu}_{i,j}$ is the supervector corresponding to the enrollment image (i, j), $\mathbf{u}_{i,j}$ is the offset induced by specific session conditions of image (i, j), and $\mathbf{d}_i$ is the client-dependent offset. (Note that the client-dependent offset $\mathbf{d}_i$ in equation (2.19) is free from session variability, unlike the $\mathbf{d}_i$ in equation (2.18).) The session-dependent offset, $\mathbf{u}_{i,j}$ can be expressed as:

$$\mathbf{u}_{i,j} = \mathbf{u}\boldsymbol{x}_{i,j} \ , \tag{2.20}$$

where, $\mathbf{u}$ is a low-dimensional matrix modeling the session-variability, and $\boldsymbol{x}_{i,j} \sim \mathcal{N}(0, \boldsymbol{I})$ is a latent variable corresponding to the session-variability. Similarly, the client-dependent offset, $\mathbf{d}_i$, can be represented as

$$\mathbf{d}_i = \mathbf{d}\boldsymbol{z}_i \ , \tag{2.21}$$

where $\mathbf{d}$ is a diagonal matrix derived from the diagonal variances of the UBM, and $\boldsymbol{z}_i \sim \mathcal{N}(0, \mathcal{I})$ is a client-specific latent random variable. The subspace $\mathbf{u}$ is estimated using an expectation-maximization (EM) algorithm. For details of the ISV technique for face-verification, refer to the works of (Wallace et al., 2011) and (Vogt and Sridharan, 2008).

When enrolling a new client, i, using a set of enrollment images (indexed by j), the latent variables $\boldsymbol{x}_{i,j}$ and $\boldsymbol{z}_i$ are estimated from the enrollment images, and finally, the client-specific supervector, $\boldsymbol{c}_i$, is computed as:

$$\boldsymbol{c}_i = \mathbf{m} + \mathbf{d}\boldsymbol{z}_i \ . \tag{2.22}$$

Thus, a single supervector, $\boldsymbol{c}_i$, is stored for each enrolled client.

Given a probe image, $\mathbb{P}$, claiming identity, i, the classification score for the probe is computed as the log-likelihood ratio $LLR(\mathbb{P}, \boldsymbol{c}_i)$ as follows:

$$LLR(\mathbb{P}, \boldsymbol{c}_i) = Q(\mathbb{P}|\boldsymbol{c}_i) - Q(\mathbb{P}|UBM) \ , \tag{2.23}$$

where, $Q(\mathbb{P}|\boldsymbol{c}_i)$ gives the log of the likelihood of the probe $\mathbb{P}$ being generated by the model $\boldsymbol{c}_i$, and similarly, $Q(\mathbb{P}|UBM)$ gives the log of the likelihood of the probe $\mathbb{P}$ being generated by the UBM. In practice, we use the linear scoring method by (Glembek et al., 2009) to compute the score. This faster scoring method is derived as a first order Taylor series approximation of the normal log-likelihood ratio. The probe is accepted if the resulting score is higher than a preset threshold.

2.3.4 ROC-SDK from Rank One Computing

This FR product, from Rank-One Computing, has been included in our study as it is one of the best performing COTS FR products today (Grother, Ngan, and Hanaoka, 2017). In particular, Rank-One Computing have demonstrated a FR performance of 92% (@ false reject rate = 1%) on the LFW dataset (G. B. Huang et al., 2007), and of 98% (@ false reject rate = 1%) on the NIST Special Database 32: Multiple Encounter Dataset (MEDS) (Watson, 2010). The software development kit (SDK) distributed by the company includes two FR methods – ROCFR (a pose-independent FR system), and, ROCID (a FR system for images captured under controlled conditions). Here, we have tested the ROCFR method from the SDK (version 1.9). This method represents every face-image with a 144-byte template. The SDK provides functions for populating a gallery with enrollment templates, and for comparing probe-templates to the gallery of previously enrolled templates.

2.4 Presentation Attack Detection (PAD)

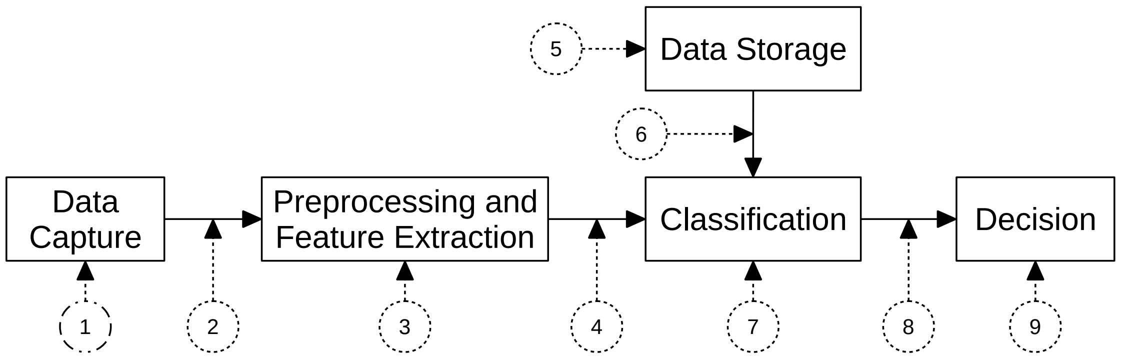


Figure 2.11 – Potential points of attack in a biometric system, as defined in (ISO/IEC DIS 30107-1, 2016). Attacks at point 1 are called presentation attacks.

Face recognition systems should not only have very high accuracy, but should also be robust to attacks. In general, a biometric recognition system can be attacked at several points as shown in figure 2.11. Attack on the biometric sensor (point 1 in figure 2.11) are called direct attacks or **presentation attacks** (**PA**s). Attacks at point 2 to 9 in figure 2.11 are called indirect attacks. An overview of indirect attacks can be found in (Gomez-Barrero et al., 2013). Countermeasures to indirect attacks can be implemented by securing the hardware, the software, and the communication channel components of the biometric system. Countermeasures to such attacks are topics related to classical cybersecurity problems, and are not specific to biometrics (Hernandez-Ortega et al., 2019). The focus of this work is on PAs as they are the only family of attacks on a biometrics system that are related to biometrics.

A face PA is said to have occurred when a face biometric-sample is presented to the camera of a FR system "with the intention of interfering with the operation of biometric recognition" (ISO/IEC DIS 30107-1, 2016). For example, person A may attack a FR system by claiming to be an enrolled client, B, and presenting a printed photo of the person B to the camera. Due to the advancement of digital technology and increasing use of social networks, it is fairly easy for attackers to gain access to face photo or videos of a person with minimal effort. The photos can be used to create PAs and fraudulently access a biometric system (Newman, 2016). Once the face biometric trait of the victim is obtained, there are several ways to create a PA. The example that was given earlier where the attackers use printed photos are often called print attacks. Other examples of PAs are: digital photos or videos displayed on an electronic screen, face sketches, masks, make-up, and surgery (Hernandez-Ortega et al., 2019; Marcel et al., 2019). The medium (or support) used to create the PA is called the **presentation attack instrument** (**PAI**) (ISO/IEC DIS 30107-1, 2016).

The challenge in **presentation attack detection** (**PAD**) (also referred as **liveness detection** or **anti-spoofing**) is to develop countermeasures to PAs, that is, to be able to identify whether the presented biometric sample is a ***bona fide*** sample (*i.e.*, a live sample), or a PA. PAD systems can be categorized in several ways:

- **Frame-based versus video-based:** some PAD systems only use a single image to classify face samples. These PAD systems can quickly output a decision after a snapshot of the presented sample is taken. On the other hand, some PAD systems require a video recording of certain length to classify the samples. These PAD systems rely on temporal cues such as eye blinking (G. Pan, L. Sun, and Z. Wu, 2008) and small involuntary movements (micro-movements) of parts of face and head (Jee, Jung, and Yoo, 2006; Anjos and Marcel, 2011) to work.

- **Visible-light versus extended-range imagery:** PAD systems can also be categorized based on their imaging sensors. Many PAD systems use the visible range of the electromagnetic spectrum – approximately 380 to 750 nm. The advantage of these PAD methods is that they can be deployed in many devices that already feature a visible-light camera. Other PAD methods use extended-range (ER) imagery technologies such as near-infrared (NIR), short-wave infrared (SWIR), and thermal imaging. These PAD methods require special hardware. However, many new smartphones such as the Samsung Galaxy S9[7] feature an NIR camera sensor. Therefore, the application of PAD systems using ER imagery is increasing (Bhattacharjee et al., 2019). Most PAs are designed to mimic the human face under visible range illumination. Their characteristics under ER illumination, however, differs from that of human faces. These differences make ER imagery attractive for PAD applications.

- **Challenge-Response:** Some PAD systems require human interaction. For example the PAD system may require the user to smile or blink (G. Pan, L. Sun, and Z. Wu, 2008). This requirement can also be used as a measure to categorize PAD systems. Many existing public face PAD datasets are not collected with the development of challenge response PAD methods in mind. Hence, these PAD methods are not as developed as other PAD methods.

The PAD systems which only use biometric cues without requiring additional hardware or human interaction, especially the frame-based systems, are more developed because they can be easily integrated with many existing biometric systems. These systems use cues such as the skin texture (Chingovska, Anjos, and Marcel, 2012) and image distortion and quality analysis (Galbally, Marcel, and Fierrez, 2014; Wen, H. Han, and Jain, 2015) to detect the PAs.

State-of-the-art face PAD methods relying on frame-based approaches use deep convolutional neural networks (CNN) as end-to-end PAD systems. These systems accept as input a face image and output a score that represents the likelihood of the image being a presentation attack. Since it has been shown that the CNN-based systems outperform the traditional handcrafted PAD systems (Z. Boulkenafet et al., 2017; Atoum et al., 2017; Almeida, 2018; George and Marcel, 2019), I will consider only CNN-based baselines in this work.

[7] See: https://en.wikipedia.org/wiki/Samsung_Galaxy_S9. The NIR sensor is labeled as an iris scanner.

2.4.1 AlexNet for Face PAD

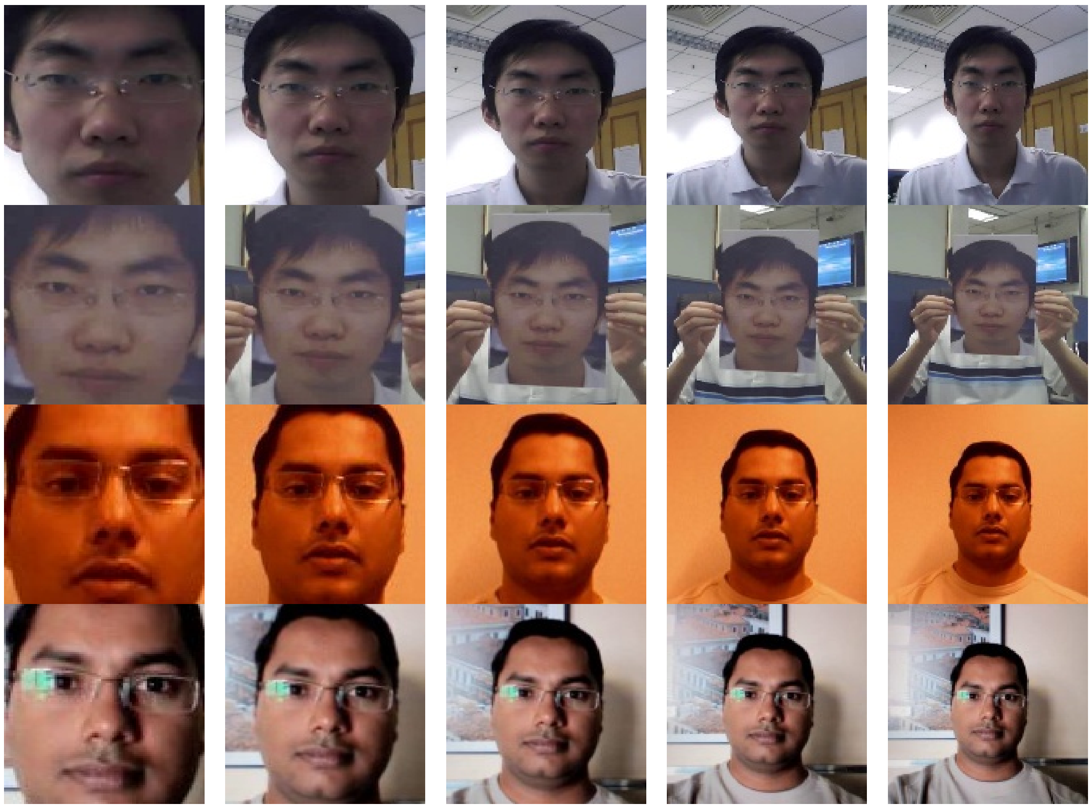


Figure 2.12 – Different face scales (cropping and normalization schemes) to be used as input to face PAD CNNs are shown. Figure from (J. Yang, Lei, and S. Z. Li, 2014). All images are 128 × 128 pixel images and in each column, faces are cropped and normalized to the same scale. At the lowest scale, scale 1 (the leftmost column), faces are cropped and normalized in a way to have minimal background visible in the normalized image. At the highest scale, scale 5 (the rightmost column), faces are much smaller but more background is visible. The top two row images are from the CASIA-FASD dataset and the bottom two row images are from the Replay-Attack dataset (see section 2.7 on page 59 for details of the datasets). In the two rows of each dataset, the upper row contains images of a BF sample and the lower row images of a PA sample.

J. Yang, Lei, and S. Z. Li (2014) propose to use CNNs for face PAD because CNNs have shown promising results in other computer vision tasks such as (Krizhevsky, Sutskever, and G. E. Hinton, 2012). They use the same AlexNet architecture as proposed in (Krizhevsky, Sutskever, and G. E. Hinton, 2012) (see section 2.2.2 on page 14 for details of the AlexNet architecture) and the input is normalized face images (see section 2.3.1 on page 28 on face normalization). The performance of the network is tested with the input being one to three frames of face images from a video recording. Also, the performance of the network is tested with different face scales in cropped and normalized face images. Five different face scales are tested which are shown in figure 2.12. At the lowest scale (scale 1), faces are tightly cropped and no background is visible, and at the highest scale (scale 5), faces are smaller with a lot of visible background. The authors argue that the background information of images is useful for detecting the

PAs. However, as we can see in figure 2.12, in the Replay-Attack dataset PA images, no real background is visible while in the CASIA-FASD dataset PA images, the real background and border of the PAs is visible. Therefore, using background information to detect PAs, can introduce bias in our PAD systems. For example, if a face PAD CNN is trained on CASIA-FASD with face images of scale 5 (see figure 2.12), it may use only visible borders as cues for detecting PAs. Then, this CNN would not generalize to other PAs where the border of the attacks are not visible. The evaluations in (J. Yang, Lei, and S. Z. Li, 2014) show improvements in terms of both intra-dataset and cross-dataset evaluations compared to hand-crafted features.

2.4.2 MSU-Patch

Table 2.9 – Architecture details of MSU-Patch by (Atoum et al., 2017). The input to the network are 96 × 96 face image patches that are extracted from original size face image. See section 2.1 on page 9 for details of the components.

Layer (type)	Details	Output Shape	Number of Parameters
Conv-1 (Conv2D)	filters=50, kernel_size=(5, 5), strides=(1, 1), padding=same	(96, 96, 50)	3,750
BN-1 (BatchNormalization)		(96, 96, 50)	150
ReLU-1 (Activation)	activation=relu	(96, 96, 50)	0
MaxPool-1 (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=same	(48, 48, 50)	0
Conv-2 (Conv2D)	filters=100, kernel_size=(3, 3), strides=(1, 1), padding=same	(48, 48, 100)	45,000
BN-2 (BatchNormalization)		(48, 48, 100)	300
ReLU-2 (Activation)	activation=relu	(48, 48, 100)	0
MaxPool-2 (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=same	(24, 24, 100)	0
Conv-3 (Conv2D)	filters=150, kernel_size=(3, 3), strides=(1, 1), padding=same	(24, 24, 150)	135,000
BN-3 (BatchNormalization)		(24, 24, 150)	450
ReLU-3 (Activation)	activation=relu	(24, 24, 150)	0
MaxPool-3 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=same	(12, 12, 150)	0
Conv-4 (Conv2D)	filters=200, kernel_size=(3, 3), strides=(1, 1), padding=same	(12, 12, 200)	270,000
BN-4 (BatchNormalization)		(12, 12, 200)	600
ReLU-4 (Activation)	activation=relu	(12, 12, 200)	0
MaxPool-4 (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=same	(6, 6, 200)	0
Conv-5 (Conv2D)	filters=250, kernel_size=(3, 3), strides=(1, 1), padding=same	(6, 6, 250)	450,000
BN-5 (BatchNormalization)		(6, 6, 250)	750
ReLU-5 (Activation)	activation=relu	(6, 6, 250)	0
MaxPool-5 (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=same	(3, 3, 250)	0
Flatten (Flatten)		(2250)	0
FC-1 (Dense)	units=1000	(1000)	2,250,000
BN-6 (BatchNormalization)		(1000)	3,000
ReLU-6 (Activation)	activation=relu	(1000)	0
Dropout (Dropout)	drop_rate=0.5	(1000)	0
FC-2 (Dense)	units=400	(400)	400,000
BN-7 (BatchNormalization)		(400)	1,200
ReLU-7 (Activation)	activation=relu	(400)	0
FC-3 (Dense)	units=2, activation=softmax	(2)	802
Model	Parameters: total=3,561,002, trainable=3,556,702		

In (Atoum et al., 2017), two CNNs are used. One takes rectangular patches (96 × 96 pixels) of face images as input and classifies each sample as *bona fide* or presentation attack. The other network takes the whole face image as input and outputs an estimated depth of the image. The depth maps for *bona fide* samples are extracted using a commercial software and are used as labels. For presentation attacks a flat depth map is used as labels because all the print and video replay attacks are expected to not have any depth. Later, the depth maps are fed to an SVM classifier for PAD.

In this work, we only implement the patch-based CNN of (Atoum et al., 2017)[8]. The MSU-Patch works on face images in their original resolution (without any resizing) and patches of 96×96 pixels are extracted from these images. (Atoum et al., 2017) argue that resizing the face images could lead to information loss. However, this patch size is very specific to the datasets that were used in (Atoum et al., 2017). Evaluation of this method on other PAD datasets is discussed in this work. The details of the CNN architecture is outlined in table 2.9. The MSU-Patch architecture is made out of five convolutional layers (Conv2D, BatchNormalization, ReLU, and MaxPooling2D) and three fully-connected layers (Dense, BatchNormalization, ReLU or softmax). The first convolutional layer uses 5×5 kernel sizes while the remaining convolutional layers use 3×3 kernel sizes. Batch normalization is used throughout the architecture.

2.4.3 Deep Pixel-wise Binary Supervision (DeepPixBiS)

George and Marcel (2019) show that training a CNN to predict correct depth maps for *bona fide* samples and a flat depth map for presentation attack samples is not necessary as it was done in (Atoum et al., 2017). Instead, they train a CNN that outputs maps (similar to depth maps) but these maps only output the probability of each *pixel* of the output map being an attack or not. This approach is similar to a patch-based CNN but the network takes as input the whole face image and outputs decisions for all parts of the face image at once. The final probability of a face image being a PA is calculated by averaging the output map probabilities.

Table 2.10 – Architecture details of DeepPixBiS by (George and Marcel, 2019). The input to the network are 224×224 normalized color face images. See sections 2.1 and 2.2.8 on pages 9 and 21 and tables 2.8 and A.9 to A.11 on pages 24 and 132 for details of the components.

Layer (type)	Details	Output Shape	Number of Parameters
conv0_pad (ZeroPadding2D)	padding=((3, 3), (3, 3))	(230, 230, 3)	0
conv0 (Conv2D)	filters=96, kernel_size=(7, 7), strides=(2, 2), padding=valid	(112, 112, 96)	14,112
norm0 (BatchNormalization)		(112, 112, 96)	384
relu0 (Activation)	activation=relu	(112, 112, 96)	0
pool0_pad (ZeroPadding2D)	padding=((1, 1), (1, 1))	(114, 114, 96)	0
pool0 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(56, 56, 96)	0
dense_block_1 (DenseBlock)	layers=6, growth_rate=48, bottleneck=True, dropout_rate=0	(56, 56, 384)	756,288
transition_block_1 (TransitionBlock)	filters=192	(28, 28, 192)	75,264
dense_block_2 (DenseBlock)	layers=12, growth_rate=48, bottleneck=True, dropout_rate=0	(28, 28, 768)	2,077,056
transition_block_2 (TransitionBlock)	filters=384	(14, 14, 384)	297,984
dec (Conv2D)	filters=1, kernel_size=(1, 1), strides=(1, 1), padding=valid	(14, 14, 1)	385
Pixel_Logits_Flatten (Flatten)		(196)	0
activation (Activation)	activation=sigmoid	(196)	0
Model	Parameters: total=3,221,473, trainable=3,198,529		

The details of the CNN architecture used in the experiments, DeepPixBiS, are shown in figure 2.13 and table 2.10. The layers from *conv0_pad* to *transition_block_2* are taken from the DenseNet-161 architecture (see table 2.8 on page 24 for details of the DenseNet-161 architecture). The architecture's weights are initialized with the weights of DenseNet-161 trained on ImageNet for the layers that are common with DenseNet-161 (see table 2.8 on page 24). The

[8]Since neither the source-code nor the trained model for the CNN proposed by (Atoum et al., 2017) is publicly available, we implemented the method using the information that was shared in the paper. In particular, the authors report 2.5%EER and 1.25%$HTER$ for the patch-based CNN on the REPLAY-ATTACK dataset. Our implementation of the work achieved 2.0%EER and 2.5%$HTER$ on the same dataset.

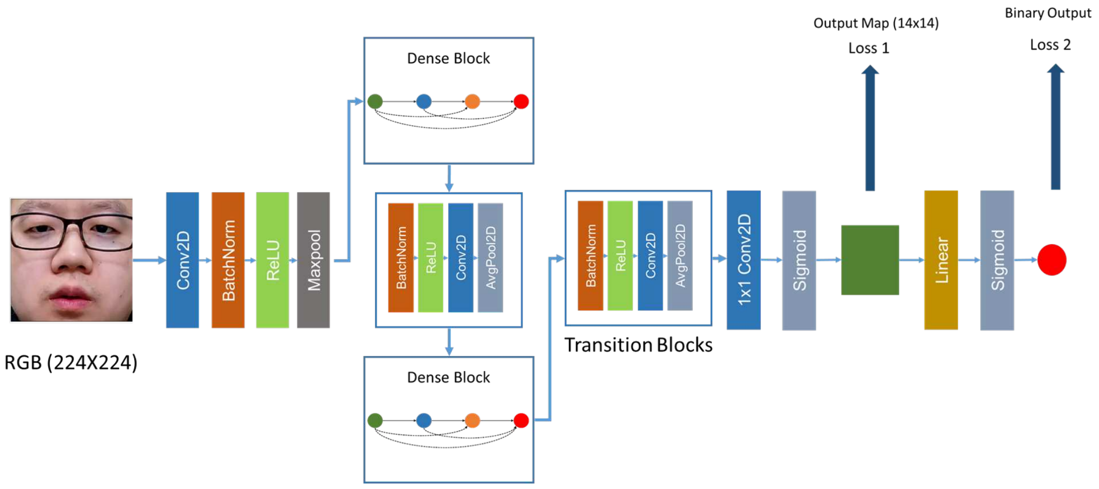


Figure 2.13 – The DeepPixBiS architecture proposed by (George and Marcel, 2019). The network starts with a convolution layer followed by two repetitions of a dense block and a transition block. These layers are the same initial layers of DenseNet-161. Output of the last transition block is given to a 1×1 convolution that outputs $14 \times 14 \times 1$ decision maps, i.e., a decision for each pixel of the feature maps. At test time, these pixel-wise decisions are averaged to report a final decision for the face image. At training time, these pixel-wise feature maps are given to a fully connected layer to make a single decision for each image. This fully connected layer is not used at test time. Figure from (George and Marcel, 2019).

rest of the weights are initialized randomly.

2.5 Domain Adaptation in Face PAD

Most machine learning models work under the assumption that the distribution of data does not change between training and evaluation data. However, it is often the case in machine learning that the distribution of the data that a model is evaluated on (target data) is different from the distribution of the data the model was trained on (source data). This change in the distribution can lead to poor performance of the model on test data (S. J. Pan and Q. Yang, 2009; Quionero-Candela et al., 2009). This change in the distribution is often called **domain shift** or **covariate shift** and solutions to this problem are called **domain adaptation** or **domain generalization** methods (S. J. Pan and Q. Yang, 2009; Quionero-Candela et al., 2009). Domain adaptation is a particular case of **transfer learning** (S. J. Pan and Q. Yang, 2009).

As we shall see in chapter 4, current face PAD systems show poor classification performance in cross-dataset evaluation scenarios. Cross-dataset evaluation scenarios represent real-world applications where the performance of a PAD system trained on a *source* dataset is evaluated on a *target* dataset. The two datasets represent two different data distributions and the low performance of PAD systems in cross-dataset scenarios may be attributed to the domain shift present between these datasets (Storkey, 2009). The domain shift may be caused due to several factors. For example in face PAD, the camera to capture face images may be different between datasets or the lighting conditions may be different. If the distribution of the target dataset is somewhat known, *i.e.,* some limited training data from the target dataset is available, then domain adaptation methods can be used to improve the performance of the model on the target dataset. Otherwise, domain generalization methods may be used which work under the assumption that no training data from the target dataset is available. Domain generalization methods result in models that are invariant to domain shifts[9].

In this section, first, the formal definitions of transfer learning and other related terms are given in section 2.5.1. Some examples of domain shift in a few visual classification problems and also in face PAD are shown in section 2.5.2. Next, two popular methods of domain adaptation and domain generalization are presented in section 2.5.3. Finally, several face PAD methods that use domain adaptation and generalization methods are detailed in section 2.5.4.

2.5.1 Terms and Definitions

Using the notation used by S. J. Pan and Q. Yang (2009), transfer learning is defined as follows. Assume a **covariate** (input) random variable, X, (with marginal probability distribution of $P(X)$) is input to a model with the objective predictive function, f, that predicts Y (the **predicted** random variable):

$$Y = f(X;\theta) \tag{2.24}$$

[9]Use of domain adaptation and domain generalization methods in face PAD will be investigated in chapters 5 and 6.

where θ represents the set of parameters of the model[10]. θ is usually learned using a training dataset consisting of pairs of $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$ samples where $\boldsymbol{x}_i \in \mathcal{X}$ and $\boldsymbol{y}_i \in \mathcal{Y}$.

The following pairs make up a **domain**, $\mathcal{D}$, and a **task**, $\mathcal{T}$:

$$\mathcal{D} = \{\mathcal{X}, P(X)\} \tag{2.25}$$

$$\mathcal{T} = \{\mathcal{Y}, f(.)\} \tag{2.26}$$

when any item of the pairs change, we say the domain or the task is changed. For example, FR and face PAD using visual light face images are two different tasks ($\mathcal{Y}$ and $f(.)$ are different) in the same domain but FR using visual light images and FR using infrared images are the same task in two different domains (at least $\mathcal{X}$ is different).

Transfer learning uses knowledge from a **source** domain and a **source** task ($\mathcal{D}_S$ and $\mathcal{T}_S$) to improve the predictive function, $f_T(.)$, on a **target** domain and task ($\mathcal{D}_T$ and $\mathcal{T}_T$) where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.

Domain adaptation is one case of transfer learning where the tasks remain the same between source and target but the domains change, *i.e.*, $\mathcal{T}_S = \mathcal{T}_T$ but $\mathcal{D}_S \neq \mathcal{D}_T$. The change in domains between source and target is referred as **domain shift**. Usually, the domains are different because the marginal probability distribution of the covariate variable changes between the source and target domains, $P_S(X) \neq P_T(X)$, which is called **covariate shift**.

Domain adaptation methods usually work under the assumption that abundant training data (usually labeled) is available in the source domain and limited training data is available in the target domain. The training data in the target domain may not be labeled. Depending on the need of labels in the target domain, domain adaptation methods may be categorized as *supervised*, *semi-supervised*, or *unsupervised*. Note that in domain adaptation, only the performance of the model on the target domain is important and $f_T(.)$ is not necessarily applicable on the samples from the source domain.

Domain generalization is similar to domain adaptation. However, in domain generalization the target domain is usually *unseen*, *i.e.*, no training data is available from the target domain. Instead, training data from *multiple source domains* is available. Domain generalization methods result in one model that performs well on the source domains and on the unseen target domain. In other words, the resulting model is *invariant* to domain shifts present between datasets.

2.5.2 Examples of Domain Shift

The covariate (input) variable may change due to changes of the **underlying factors** that explain the data. For example, in images, these factors can be illumination, saturation, pose,

[10] $f(X;\theta)$ can be written as $P(Y|X)$ from a probabilistic viewpoint as well.

image quality, camera device, and so on. Some examples of domain shift in visual classification problems are shown in figure 2.14. In figure 2.14.a, we can see that images of objects like bicycles and laptops can change drastically between datasets. In the Amazon dataset, all images have a white background and uniform illumination. However, the background and illumination are not kept constant in the other three datasets in figure 2.14.a. A model trained for object classification on the Amazon dataset may have a degraded performance when tested on other datasets. In figure 2.14.b, we can see that the size, thickness, color, and background of digits vary between the digits datasets. These factors can affect the performance of a digit classification model. In figure 2.14.c, we can see that the LFW dataset contains face images of adults, the BCS dataset contains face images of babies, and the CUFS dataset contains images of face sketches. Performance of a model trained for face recognition on any of these three datasets, may degrade significantly when tested on another of these datasets. These are examples of *domain shift* that can affect the performance of the models.

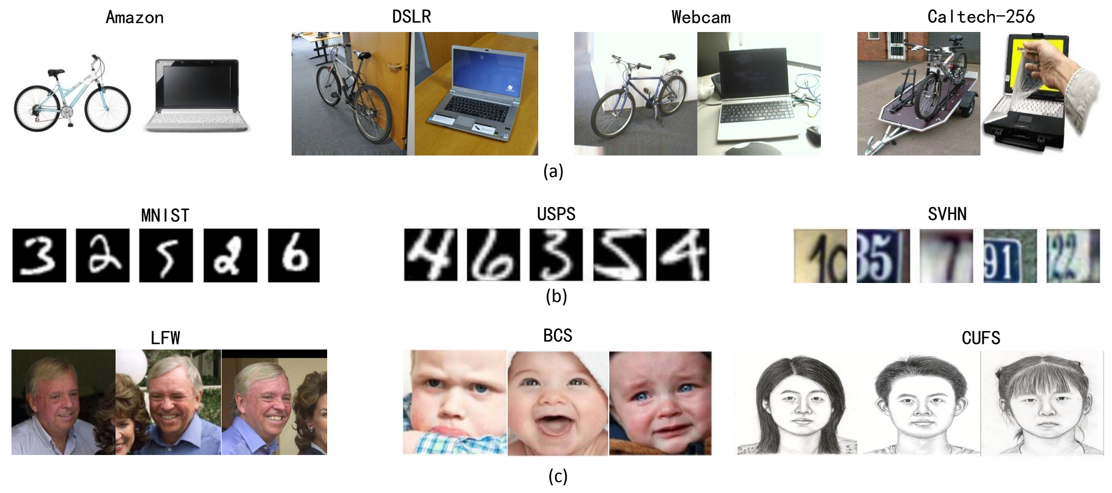


Figure 2.14 – Examples of domain shifts between datasets in a variety of tasks. In (a), images from two classes (bicycle and laptop) are shown from four datasets of object classification. In (b), various digits are shown from three different datasets of digit classification. In (c), face images are shown from three datasets for face recognition. In each task, a dataset can contain biases or have a different domain than other datasets in the same task. Figure from (Wang and Deng, 2018).

Examples of domain shift in face PAD can be seen in figure 2.15 where the samples from each class can change drastically *between datasets*. The biometric sensor (the camera device, in this case), the PAI used to create the attack, illumination, identity, pose of the subject, distance of the subject to the camera, and many other factors can cause domain shifts between datasets.

2.5.3 Domain Adaptation and Domain Generalization Methods

There exist many domain adaptation and domain generalization methods (V. M. Patel et al., 2015) and many have been developed for deep learning based models (Wang and Deng, 2018). Some of these methods have been applied to the problem of face PAD as well (H. Li, P. He,

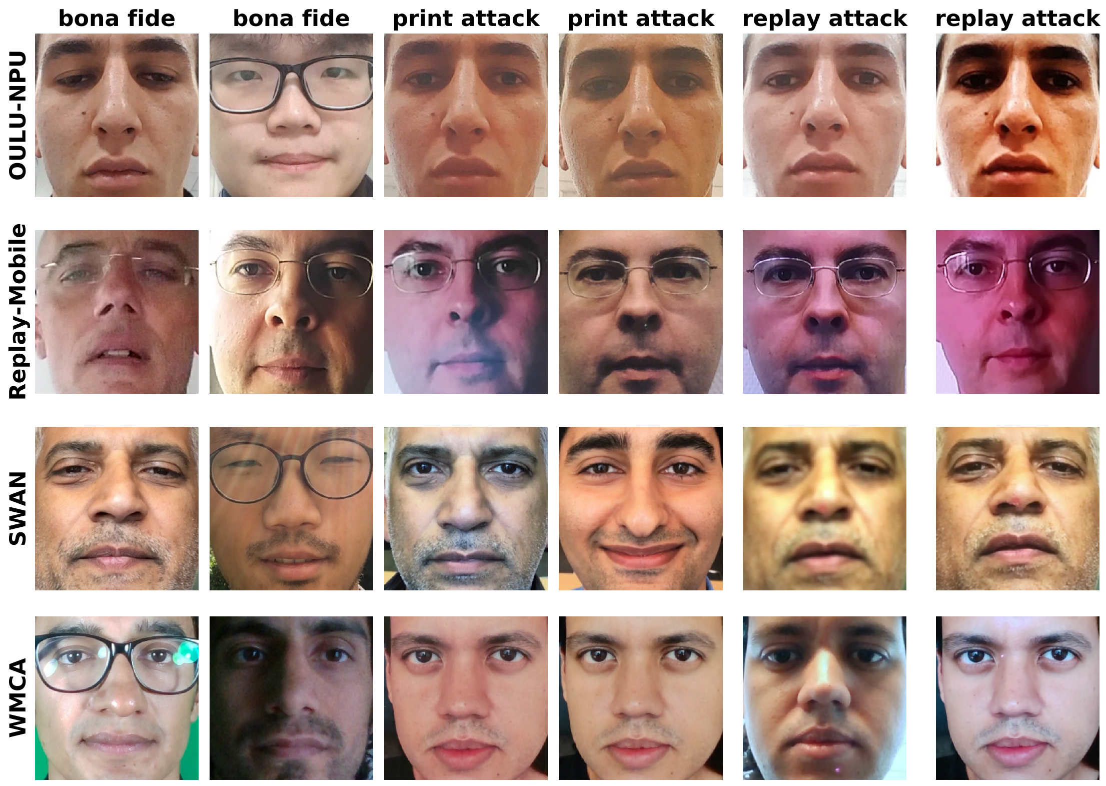


Figure 2.15 – Examples of domain shift between datasets in face PAD. The samples of each row belong to the same dataset. The datasets from top to bottom are OULU-NPU, Replay-Mobile, SWAN, and WMCA. The first two columns are *bona fide* samples, the second two columns are print attacks, and the last two columns are replay attacks. Samples from each class can change drastically between datasets. The identity of the person, pose, illumination, camera device, PAI used to create the attack, and other factors are the cause of domain shift between these datasets.

et al., 2018; H. Li, W. Li, et al., 2018; Shao et al., 2019; Xiaoguang Tu, J. Zhao, et al., 2019; Zhou et al., 2019). Two commonly used classes of domain adaptation methods are:

- maximum mean discrepancy (MMD) based methods, and,
- adversarial based methods.

In both methods, the goal is to match the distributions of the source and target data in a learned feature space while maintaining discriminative features for the prediction task. If the features have similar distributions, a classifier learned on features of the source samples can also be used to classify the target samples. The details of these methods are given below and the related work on face PAD is presented in section 2.5.4.

2.5.3.1 Maximum Mean Discrepancy (MMD)

Maximum Mean Discrepancy (MMD) is a statistical test to determine whether two distributions are different. Given samples that are drawn from two distributions, MMD computes the distance between the *mean embedding* of their samples (Gretton et al., 2012). Given two i.i.d. observations X and Y (in space $\mathcal{X}$) drawn from distributions P and Q, respectively, MMD is given as

$$MMD(P,Q) = \|\mathbb{E}_{X\sim P}[\varphi(X)] - \mathbb{E}_{Y\sim Q}[\varphi(Y)]\|_{\mathcal{H}}. \tag{2.27}$$

where $\varphi : \mathcal{X} \to \mathcal{H}$ is a function that maps samples to embeddings in a general **reproducing kernel Hilbert space (RKHS)**[11], $\mathcal{H}$. If the dot product of two embeddings can be computed using a kernel, k:

$$k(x,y) = \langle \varphi(x), \varphi(y) \rangle_{\mathcal{H}} \tag{2.28}$$

MMD^2 can be computed by applying the kernel trick[12]:

$$\begin{align}
MMD^2(P,Q) &= \|\mathbb{E}_{X\sim P}\varphi(X) - \mathbb{E}_{Y\sim Q}\varphi(Y)\|^2_{\mathcal{H}} \tag{2.29}\\
&= \langle \mathbb{E}_{X\sim P}\varphi(X), \mathbb{E}_{X'\sim P}\varphi(X')\rangle_{\mathcal{H}} + \langle \mathbb{E}_{Y\sim Q}\varphi(Y), \mathbb{E}_{Y'\sim Q}\varphi(Y')\rangle_{\mathcal{H}} - 2\langle \mathbb{E}_{X\sim P}\varphi(X), \mathbb{E}_{Y\sim Q}\varphi(Y)\rangle_{\mathcal{H}} \tag{2.30}\\
&= \mathbb{E}_{X,X'\sim P}k(X,X') + \mathbb{E}_{Y,Y'\sim Q}k(Y,Y') - 2\mathbb{E}_{X\sim P,Y\sim Q}k(X,Y) \tag{2.31}
\end{align}$$

Gaussian kernels are commonly used in practice. For Gaussian kernels, as well as many other kernels, MMD is zero if and only if the two distributions are identical (Gretton et al., 2012).

For domain adaptation, MMD is used as a loss-function over features. In other words, a domain adaptation method may be designed where the objective is to minimize MMD between features of source and target domains. As mentioned before, if the samples of both domains can be mapped into a feature space where the distributions are similar between source and target samples. Then, a classifier trained on features of the source samples may be used to classify the features of the target sample as well. This can be done in a supervised or an unsupervised manner. In the unsupervised case, MMD is computed on samples from all classes of the source and target domains and is minimized. In the supervised case, MMD is computed separately for each class and its average over all classes is minimized.[13]

MMD-based domain generalization methods exist as well. The trick is to minimize MMD between features of multiple source domains. Then, the resulting feature space is said to be invariant to domain shifts and can generalize to other unseen domains.

2.5.3.2 Adversarial Domain Adaptation

Adversarial domain adaptation methods (Tzeng et al., 2017; Ganin and Lempitsky, 2014; Tzeng et al., 2015; M.-Y. Liu and Tuzel, 2016) use the generative adversarial network (GAN) training

[11] https://en.wikipedia.org/wiki/Reproducing_kernel_Hilbert_space

[12] https://en.wikipedia.org/wiki/Kernel_method

[13] In the PAD works that are discussed in this thesis, PAD is seen as a binary classification problem where the two classes are *bona fide* and presentation attacks. This keeps the number of classes constant between datasets and domain adaptation methods can be applied.

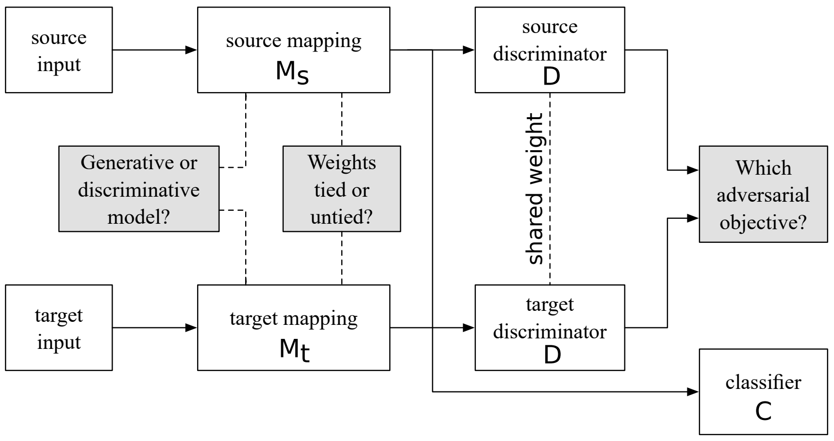


Figure 2.16 – A general framework for adversarial domain adaptation proposed by Tzeng et al. (2017). Different answers to the questions from the figure result in different adversarial domain adaptation methods. See section 2.5.3.2 for more details. M_s, M_t, C, and D in the text are referred as source mapping, target mapping, classifier, and discriminator in this figure. Both *source* and *target* discriminators are referred to as D in the text since they share weights. Figure from (Tzeng et al., 2017).

framework (Goodfellow et al., 2014) to account for the domain shift present between *source* and *target* datasets. Tzeng et al. (2017) propose a general framework for adversarial domain adaptation that subsumes most other methods as special cases. Again, the goal here is to learn a feature space where samples from the source and target domain have the same distribution. Then, a classifier trained on source samples may be used to classify the target samples. This general framework is described below.

In general, the framework involves four neural networks:

- M_s and M_t which map inputs from the source and target domains, respectively, to a common feature space;
- a classifier, C, which learns the classification task at hand by taking the features as input and outputting class probabilities; and
- a discriminator, D, trained with the objective of discriminating between source and target features.

The training of the networks is performed in two stages:

1. C is trained using a classification loss-function, such as the cross-entropy loss function, on training samples of the source dataset. M_s may be trained together with C using the same classification loss function or it may be trained as part of a generative model. Once M_s and C are trained at this stage, they are kept fixed.

2. M_t and D are trained in an adversarial manner as if M_t is the generator and D is the discriminator. The features extracted by M_t are considered *fake* samples and the features that are extracted by M_s are considered *real* samples. D is trained to discriminate between real and fake samples and M_t is trained to output samples that are indistinguishable from real samples.

M_s and M_t may share weights and the adversarial objective may vary (see *adversarial losses* in (Tzeng et al., 2017)). The general framework is shown in figure 2.16. Depending on the answers given to the questions in the figure, a new adversarial domain adaptation method can be created.

2.5.3.3 Adversarial Discriminative Domain Adaptation (ADDA)

Tzeng et al. (2017) also propose a novel domain adaptation method based on the proposed general framework named **adversarial discriminative domain adaptation (ADDA)**. In the first stage of training, M_s and C are trained end-to-end, in a discriminative manner, on the source dataset using the cross entropy loss. In the second stage of the training, M_s is fixed while M_t and D are trained with an adversarial objective. M_s and M_t do not share weights and only M_t is initialized with the weights of M_s. Note that the labels of the target domain training samples are not needed, i.e., the method is an unsupervised domain adaptation method. After training, M_t and C are used to classify samples of the target domain. The adversarial objective for ADDA is chosen to be the same *GAN loss* function proposed by Goodfellow et al. (2014).

2.5.4 Domain Adaptation and Generalization Applied in Face PAD

H. Li, P. He, et al. (2018) propose a 3D CNN with 5 convolutional layers and 2 fully connected layers. The input to the 3D CNN consists of 8 consecutive frames of face images. H. Li, P. He, et al. (2018) consider different camera devices used in recording of face videos as different domains. Three face PAD datasets: REPLAY ATTACK (Chingovska, Anjos, and Marcel, 2012), MSU MFSD (Wen, H. Han, and Jain, 2015), and CASIA FASD (Z. Zhang et al., 2012) are used. In each dataset, several different camera devices are used in recording of face videos. The three datasets are combined to create 10 protocols. In each protocol, samples of one camera device is left out of training and samples from the rest of the camera devices are used for training. The CNN is trained to be invariant to camera models and to be able to generalize to unseen camera models. The training is done in two stages. First, the network is trained on the source datasets for face PAD using the cross entropy loss function. Then, the last convolutional layer of the network and 2 fully connected layers are further adapted to minimize MMD (see section 2.5.3.1) between the samples from different domains (camera models). The MMD is computed for *bona fide* and attacks separately. The authors report that the additional domain adaptation step brings 10% of absolute improvement in HTER on average.

H. Li, W. Li, et al. (2018) introduce an unsupervised domain adaptation method using MMD.

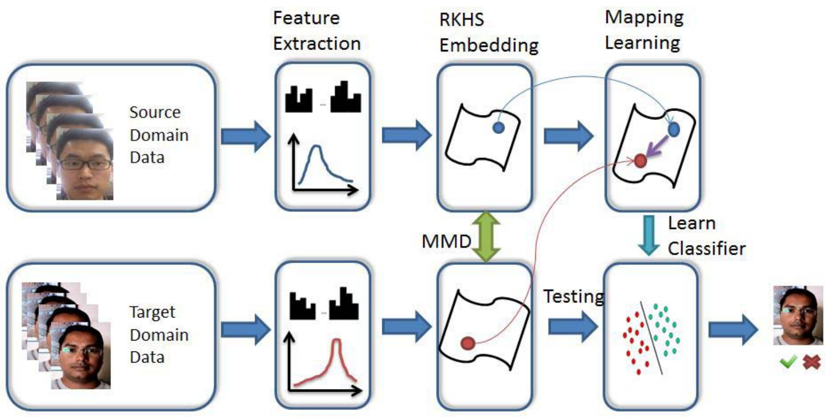


Figure 2.17 – Unsupervised domain adaptation method proposed by H. Li, W. Li, et al. (2018). Features are mapped to an RKHS embedding space and a mapping is learned to map source features to target features in this space. This mapping is learned with the objective of minimizing MMD between the source and target features in the kernel space. Once the mapping is learned, a classifier is trained on mapped source features. At test time, target features in the kernel space are tested against the trained classifier. Figure from (H. Li, W. Li, et al., 2018)

The objective is to learn a mapping that brings the distribution of source features close to the distribution of target features in a reproducing kernel Hilbert space (RKHS). The mapping is learned in a way to minimize MMD between source and target distributions in the RKHS. Once the source features are mapped, a classifier is learned on the mapped source features. At test time, given a new sample in the target domain, target features in the kernel space are labeled using the trained classifier. Figure 2.17 shows a schematic of the method. Results are shown for hand-crafted such as LBP (Chingovska, Anjos, and Marcel, 2012) and CoALBP (Nosaka, Ohkawa, and Fukui, 2011) and embeddings extracted from AlexNet (Krizhevsky, Sutskever, and G. E. Hinton, 2012) as feature vectors. The authors compare the cross-dataset test results with and without the proposed domain adaptation method. They report an absolute improvement of 24% in HTER (on average, using AlexNet embeddings) when data from both *bona fide* and presentation attacks are used to compute MMD in an unsupervised manner. When only *bona fide* data is used to compute MMD, the absolute improvement in HTER on average is reported as 14%. The datasets that are used for evaluation are REPLAY ATTACK (Chingovska, Anjos, and Marcel, 2012), MSU MFSD (Wen, H. Han, and Jain, 2015), and CASIA FASD (Z. Zhang et al., 2012). [14]

[14]Open source implementation of the methods discussed in this section are not available.

2.6 Performance Evaluation

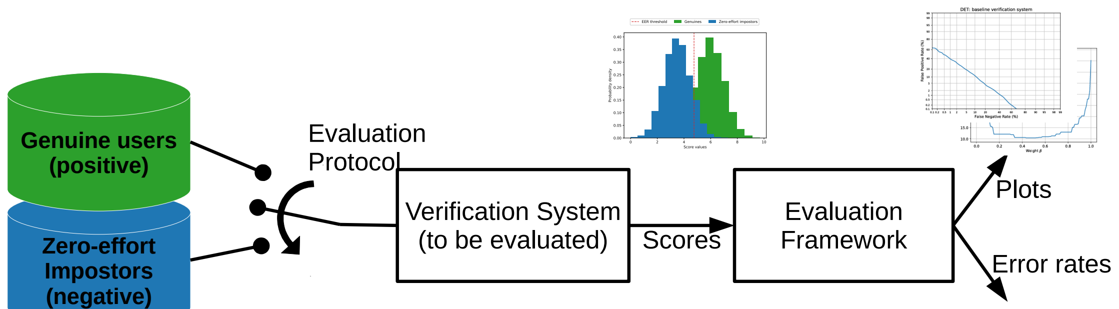


Figure 2.18 – Evaluation of a verification system with regards to its capacity to discriminate genuine samples (positives) from zero-effort impostor samples (negatives). Data samples from each class are scored by the verification system and collected scores are used to evaluate the system. Figure from (Chingovska et al., 2019).

In a biometric recognition system, recognition applies to two tasks: verification (also called authentication) and identification. In verification, the system is trying to match a biometric of a claimed identity against a pre-stored *enrollment* sample. In identification, the system is trying to identify a presented sample against known identities where their enrollment samples is present in a database. In this work, when recognition is mentioned the verification part of the task is intended. The verification problem can be seen as a binary classification problem where presentations that are being matched against the same reference identity are positive samples (genuine samples) and the presentations that are being matched against another identity are considered negative samples (zero-effort impostor, ZEI, samples). Evaluation of verification systems as a binary classification problem is done using common metrics (error rates) and plots that are designed for binary classification problems. Figure 2.18 outlines such an evaluation framework.

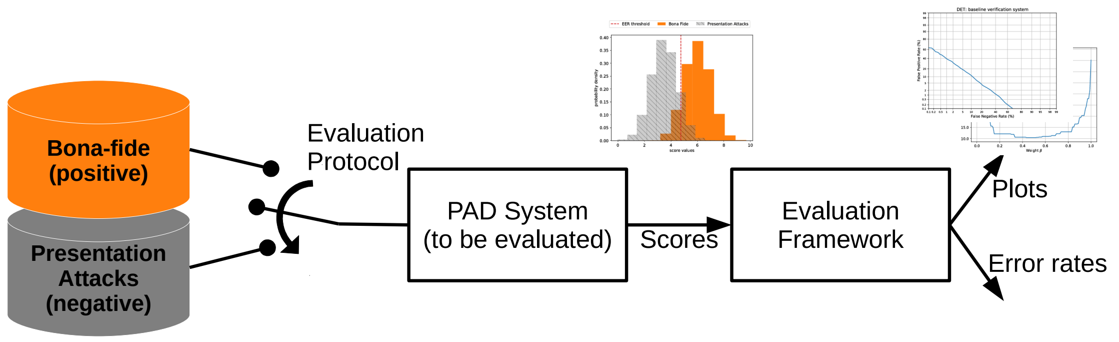


Figure 2.19 – Evaluation of a PAD system with regards to its capacity to discriminate *bona fide* samples (positives) from PA samples (negatives). Figure from (Chingovska et al., 2019).

Moreover, biometric systems are vulnerable to presentation attacks (PAs). Presentation attacks are samples that are presented to the data capture subsystem of a biometric system with the intention of interfering with the operation of the biometric system (ISO/IEC DIS 30107-1, 2016)[15]. If a sample is not a PA, it is called a *bona fide* sample. PAD systems discriminate between *bona fide* samples (positives) and PA samples (negatives). Most often PAD systems are evaluated as binary classification systems as shown in figure 2.19

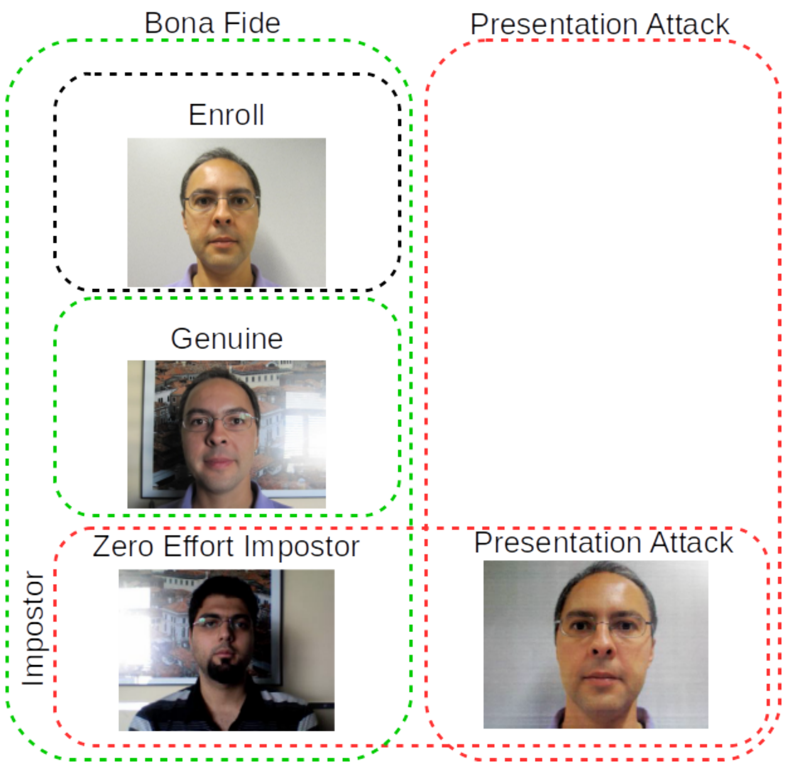


Figure 2.20 – Categorization of biometric samples in terms of a biometric recognition and a presentation attack detection (PAD) system. In terms of a PAD system, *bona fide* samples (left column) are positive samples (samples that need to be accepted) and presentation attacks (right column) are negative samples (samples that need to be rejected). In terms of a biometric recognition system, genuine samples are positive samples and impostors (both zero-effort impostors and impostor presentation attacks) are negative samples.

The categorization of biometric samples is shown in figure 2.20[16]. We can see that while ZEIs are negative samples in a biometric recognition system, they are considered as positive samples in a PAD system because they are *bona fide* samples. In real-world scenarios, a PAD system is always used alongside a biometric recognition system. Both systems operate as one larger system which is responsible for accepting genuine samples and rejecting impostor samples (ZEIs and PAs). Evaluation of this unified system can be seen as pseudo ternary classification problem (Chingovska et al., 2019) where there are two sub-classes of negatives (ZEI and PA samples) and one class of positives (genuine samples). This is demonstrated in figure 2.21. The *expected performance and spoofability* (EPS) evaluation framework (Chingovska

[15]In this work we are only interested in *impostor presentation attacks*, where an impostor is trying to gain access to the biometric system as another identity, not concealers were they try to avoid being matched by the biometric system.

[16]Note that, in this thesis, I assume enrollment samples are always *bona fide* samples which is not necessarily true.

et al., 2019; Chingovska, Anjos, and Marcel, 2014) is proposed to evaluate biometric systems when they are seen as a pseudo ternary classification problem.

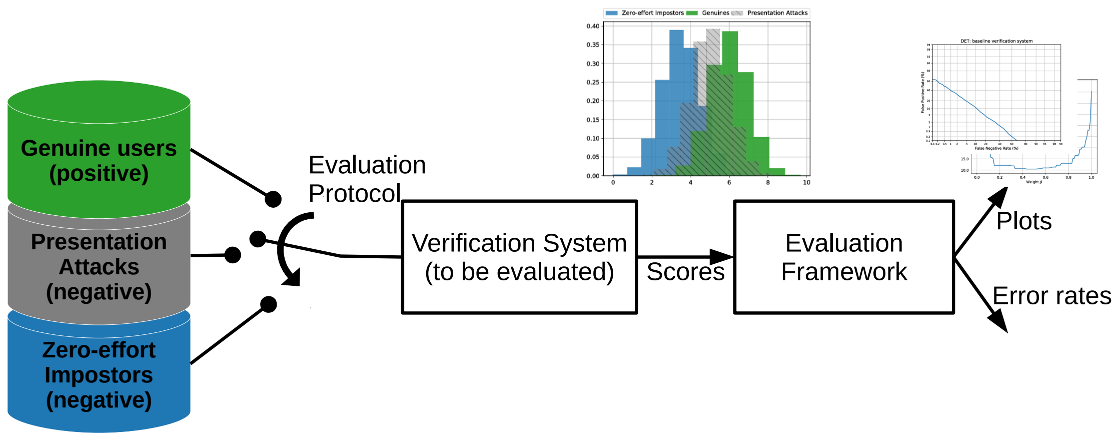


Figure 2.21 – Evaluation of a verification system with regards to its capacity to discriminate genuine samples (positives) from negative samples (ZEIs *and* PAs). Figure from (Chingovska et al., 2019).

To summarize, both biometric verification and PAD systems can be evaluated as binary classification systems (see figures 2.18 and 2.19). Moreover, biometric verification systems can also be evaluated as pseudo ternary classification systems when PAs are present in the evaluation (see figure 2.21). In the following, first, an introduction to evaluation of binary classification systems using generic terminology is given in section 2.6.1. Then, the mapping of this generic terminology to biometric verification and PAD is given in section 2.6.2. Moreover, evaluation of biometric verification systems as pseudo ternary classification systems are also presented in section 2.6.2. Finally, the datasets that are used for the experiments in this thesis are presented in section 2.7.

2.6.1 Generic Performance Evaluation

2.6.1.1 Metrics for binary classification systems

Assume a model, M, trained for binary classification, produces a score, s, for a given sample, x:

$$s = M(x) \tag{2.32}$$

to classify x as either a **positive (P)** or a **negative (N)** sample, an operating threshold, τ, is used to label x:

$$\text{label}_x = \begin{cases} \text{positive} & \text{if } s \geq \tau \\ \text{negative} & \text{if } s < \tau \end{cases} \tag{2.33}$$

Depending on if x is a positive or negative sample and its labeled as a positive and negative sample, the following outcomes are possible:

- **true positive (TP)**: when x is a positive sample and is labeled as a positive sample.
- **true negative (TN)**: when x is a negative sample and is labeled as a negative sample.
- **false positive (FP)**: when x is a negative sample and is labeled as a positive sample.
- **false negative (FN)**: when x is a positive sample and is labeled as a negative sample.

When a set of samples are scored for evaluation, the following metrics can be computed[17]:

- **sensitivity, recall, hit rate, or true positive rate (TPR)**:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR} \tag{2.34}$$

- **specificity, selectivity or true negative rate (TNR)**:

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} = 1 - \mathrm{FPR} \tag{2.35}$$

- **precision or positive predictive value (PPV)**:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = 1 - \mathrm{FDR} \tag{2.36}$$

- **negative predictive value (NPV)**:

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} = 1 - \mathrm{FOR} \tag{2.37}$$

- **miss rate or false negative rate (FNR)**:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}} = 1 - \mathrm{TPR} \tag{2.38}$$

- **fall-out or false positive rate (FPR)**:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR} \tag{2.39}$$

- **half total error rate (HTER)**:

$$\mathrm{HTER} = \frac{\mathrm{FPR}+\mathrm{FNR}}{2} \tag{2.40}$$

These metrics may be used to compare the performance of several classification systems given an operating threshold, τ, for each system. Usually, the thresholds of systems are selected using the same **criteria**. Some common criteria for selecting thresholds are:

[17] See https://en.wikipedia.org/wiki/Precision_and_recall#Definition_(classification_context) for more information

- **fixed value of FPR (or FNR)**: the threshold that results in the given FPR (or FNR) value.
- **equal error rate (EER)**: the threshold that results in equal values for FPR and FNR.
- **minimum weighted error rate (min-WER)**: the threshold that minimizes:

  $$\text{cost} * FPR + (1 - \text{cost}) * FNR \tag{2.41}$$

  where cost determines the importance that we want to give to one of the FPR or FNR metrics. When cost is 0.5, the criteria is called **minimum half total error rate (min-HTER)**.

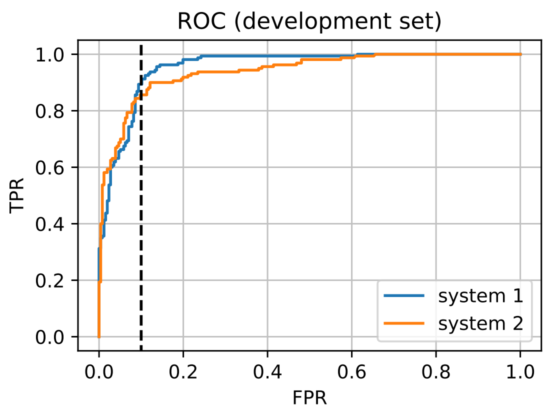
Selecting the threshold, requires access to the underlying score distribution of the evaluation. If care is not taken in the selection of the threshold, the resulting evaluation and comparison of systems may be biased. This is because in real-world applications we do not have access to the score distribution of our test samples. Therefore, it is important that thresholds are chosen *a priori* in our evaluations. Usually this is done by introducing two set of samples for evaluation: the *development set* and the *evaluation set*. The development set is used to select a threshold *a priori* and this threshold is then used to compute error rates on the evaluation set. For example, it is common to select threshold based on the EER criteria on the development set and report HTER on both development and evaluation sets[18].

2.6.1.2 Plots for binary classification systems

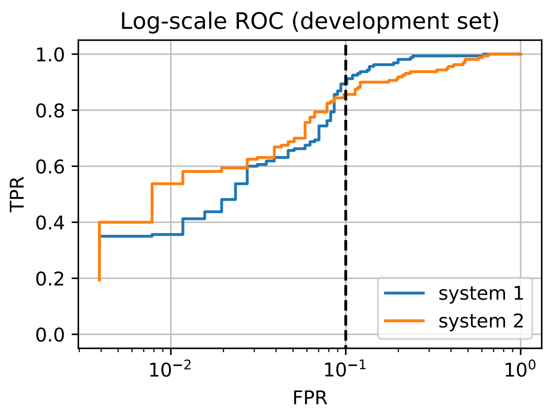
Reporting error rates such as HTER can be limiting because they correspond to one operating threshold of the systems. Moreover, the error rates are correlated. For example, decreasing the value of the operating threshold will decrease FNR and increase FPR. **Receiver operating characteristic (ROC)**[19] plots can be used to visualize the trade-off between FPR and FNR values when the operating threshold changes. Examples of ROC plots are shown in figure 2.22. The ROC plots are computed by scanning a range of possible values for the threshold and computing the corresponding FPR and TPR error rates. We can see that as the FPR decreases, TPR decreases as well. In biometrics verification evaluations, usually log-scale ROC plots are used for evaluation. In those plots, the x-axis (FPR) is shown in log-scale. Moreover, as discussed before, to avoid biased evaluations, it is often desired to compute a threshold *a priori* on a development set and use this threshold to compute error rates on an evaluation set. This is demonstrated in the ROC plots in figure 2.22 as well where thresholds are selected using a fixed value of FPR on the development set (a vertical line in the development set plots) and are used to compute error rates on the evaluation set (dots in the evaluation set plots).

Area under the curve (AUC)[20] of ROC plots are used to summarize the performance of one system using one number. Then, the performance of systems may be compared using these

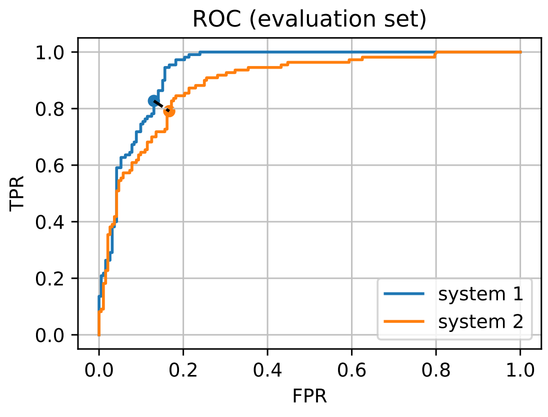
[18]It may happen that the HTER value on the development set is reported by the name of EER.
[19]https://en.wikipedia.org/wiki/Receiver_operating_characteristic
[20]https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve



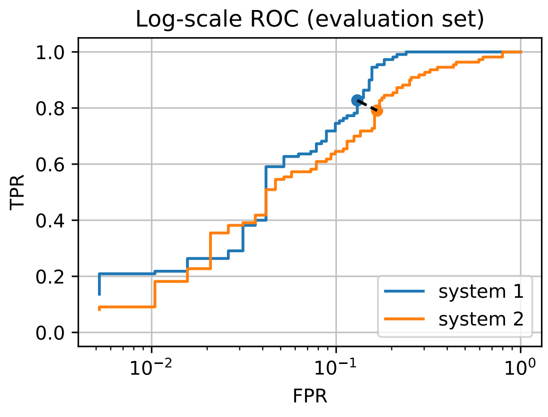
(a) – An ROC plot on the development set.



(b) – A log-scale ROC plot on the development set.



(c) – An ROC plot on the evaluation set.



(d) – A log-scale ROC plot on the evaluation set.

Figure 2.22 – Examples of receiver operating characteristic (ROC) plots evaluating two binary classification systems using the development and evaluation sets. Figures 2.22a and 2.22c show normal ROC plots and figures 2.22b and 2.22d show log-scale ROC plots. Using the development set (figures 2.22a and 2.22b) the threshold for obtaining FPR of 0.1 is calculated for each system. These thresholds correspond to the dashed black vertical lines in the development set figures. Then, using these thresholds, FPR and TPR of the systems are calculated on the evaluation set and these are shown as dots on the ROC plots of the evaluation set (figures 2.22c and 2.22d). Performance of the system 1 and 2 may be compared using these dots.

numbers. In normal ROC plots, **AUC** is a number between 0 and 1 where 1 corresponds to perfect classification performance, 0.5 corresponds to chance performance, and 0 corresponds to the worst possible performance where all samples are misclassified. However, AUC may be computed for *log-scale ROC* plots as well. **AUC of log-scale ROC** plots is a positive number that may be larger than 1 compared to the normal AUC values[21]. Higher values of *AUC of log-scale ROC* indicates higher performance of a system.

The **expected performance curve (EPC)** was introduced by S. Bengio, Mariéthoz, and Keller (2005) to account for the bias inherit in comparison of systems using ROC curves. When

[21]Because the x-axis is plotted in log-scale, the x-axis values could go to negative infinity when x is very close to 0. Therefore, the *area* under the curve of log-scale ROC plots can have values much higher than 1.

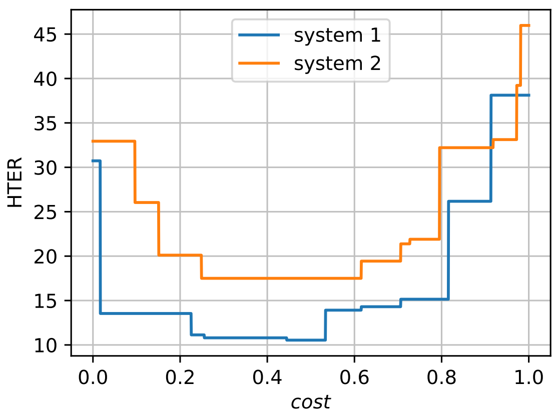


Figure 2.23 – An expected performance curve (EPC) plot evaluating two binary classification systems using the development and evaluation sets. $cost$ is from equation (2.41) (*min-WER* formula). Given a $cost$, using the scores from the development set, a threshold for each system is selected which minimizes equation (2.41). Then, thresholds are used to compute an error rate (here, HTER) on the evaluation set. The underlying evaluations are the same as in figure 2.22.

comparing systems in ROC curves by drawing a vertical (or horizontal) line in the plots and comparing the systems based on the obtained TPR (or FPR) values, we are comparing systems using *a posteriori* thresholds which leads to biased evaluations. One solution is to select thresholds on the development set and show the error rates of these thresholds on the ROC curves of the evaluation set as dots as was shown in figure 2.22. Comparing systems using these dots leads to unbiased evaluations. EPC uses the idea of *a priori* thresholds to compute unbiased curves. For computing EPC, both development and evaluation sets are *always* needed. The curve is computed in two steps. First, a set of thresholds for each system is selected on the *development set* by changing the cost from 0 to 1 in equation (2.41) (*min-WER* formula) for threshold selection. Each threshold will correspond to one cost value. Then, the thresholds are used to compute error rates (such as HTER) on the *evaluation set*. The final plot uses the searched cost values on the x-axis and computed error rates on the y-axis. An example is shown in figure 2.23. Note that any error rate such as FPR, FNR, or HTER may be reported on the y-axis.

Histogram plots may also be used to visualize the underlying score distributions as shown in figure 2.24. Although they are usually not used to compare the performance of several systems, they may be used for analyzing the performance of one system.

2.6.2 Biometrics Performance Evaluation

2.6.2.1 Biometric Verification Evaluation (Licit Scenario)

If there are no PAs present in the evaluation of biometric verification systems, usually referred as the **licit scenario**, these systems can be evaluated as binary classification systems. In this case, genuine samples are considered positive samples and ZEIs are considered negative

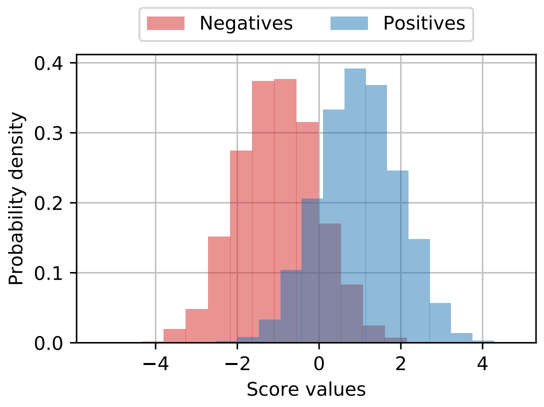


Figure 2.24 – A histogram plot of score distributions. The higher the overlap of the positive and negative scores, the higher the error rates.

samples. Then, the same metrics and plots that were introduced in section 2.6.1.1 may be used to evaluate biometric verification systems (ISO/IEC 19795-1, 2006). A mapping of the generic terms used in section 2.6.1.1 to the biometric verification evaluation terms are given in table 2.11 (second column). Besides those metrics, the following metrics are also defined in (ISO/IEC 19795-1, 2006):

- **failure-to-acquire rate (FTA)**: "proportion of verification or identification attempts for which the system fails to capture or locate an image or signal of sufficient quality" (ISO/IEC 19795-1, 2006). In other words, the ratio of samples that the system fails to output a score.
- **false reject rate (FRR)** "proportion of verification transactions with truthful claims of identity that are incorrectly denied" (ISO/IEC 19795-1, 2006)

$$FRR = FTA + FNMR * (1 - FTA) \tag{2.42}$$

- **false accept rate (FAR)** "proportion of verification transactions with wrongful claims of identity that are incorrectly confirmed" (ISO/IEC 19795-1, 2006)

$$FAR = FMR * (1 - FTA) \tag{2.43}$$

2.6.2.2 Biometric Verification Evaluation (Spoof Scenario)

If there are no ZEIs present in the evaluation of biometric verification systems, usually referred as the **spoof scenario**, these systems can be evaluated as binary classification systems. In this case, genuine samples are considered positive samples and PAs are considered negative samples. Then, the same metrics and plots that were introduced in section 2.6.1.1 may be used to evaluate biometric verification systems (ISO/IEC DIS 30107-3, 2016). A mapping of the generic terms used in section 2.6.1.1 to the biometric verification evaluation terms are given in table 2.11 (third column).

2.6.2.3 PAD Evaluation

Similarly, evaluating PAD systems in isolation as binary classifiers is possible. In this case, *bona fide* samples are considered positive samples and PAs are considered negative samples. A mapping of the generic terms used in section 2.6.1.1 to the PAD evaluation terms (ISO/IEC DIS 30107-3, 2016) are given in table 2.11 (fourth column)[22].

Table 2.11 – The mapping between the generic binary classification terms and evaluation of biometric systems when evaluated as binary classifiers. The terms that are not defined clearly in the field are not given.

Generic	**Biometric Verification licit scenario**	**Biometric Verification spoof scenario**	**PAD**
Positive	Genuine	Genuine	Bona Fide (BF)
Negative	Zero-Effort Impostor (ZEI)	Presentation Attack (PA)	Presentation Attack (PA)
True Positive (TP)	True Match (TM)	-	-
True Negative (TN)	True Non-Match (TNM)	-	-
False Positive (FP)	False Match (FM)	-	-
False Negative (FN)	False Non-Match (FNM)	-	-
True Positive Rate (TPR)	True Match Rate (TMR)	-	-
True Negative Rate (TNR)	True Non-Match Rate (TNMR)	-	-
False Positive Rate (FPR)	False Match Rate (FMR)	Impostor Attack Presentation Match Rate (IAPMR)	Attack Presentation Classification Error Rate (APCER)
False Negative Rate (FNR)	False Non-Match Rate (FNMR)	False Non-Match Rate (FNMR)	Bona fide Presentation Classification Error Rate (BPCER)
Half Total Error Rate (HTER)	Half Total Error Rate (HTER)	-	Average Classification Error Rate (ACER)

2.6.2.4 Evaluation of Biometric Systems as a Pseudo Ternary Classification Problem

As we saw in section 2.6 (figure 2.21), evaluation of biometric verification systems can be seen as a pseudo ternary classification. In this case, there is one class of positive samples: genuine, and two sub-classes of negative samples: ZEI and PA. When evaluating biometric systems under this condition, we must acknowledge that, in real-world applications, a biometric system is operated by one operating threshold *only* (Chingovska, Anjos, and Marcel, 2014; Chingovska et al., 2019). In other words, it is not realistic to evaluate a biometric system in isolated *licit* and *spoof* scenarios. Chingovska, Anjos, and Marcel (2014) have proposed a new evaluation framework called the **expected performance and spoofability (EPS) framework** which accounts for the fact that biometric systems may be presented to both ZEI and PA samples at evaluations. The framework introduces two cost parameters that determines how much importance is given to each class of sample. $\omega \in [0,1]$ is the relative importance cost of

[22]Note that APCER should be reported per PAI species according to (ISO/IEC DIS 30107-3, 2016). However, this fact is ignored in this thesis for the simplicity of evaluations.

PAs with respect to ZEIs and $\beta \in [0,1]$ is the relative importance cost of negative classes (ZEI and PAs) with respect to the positive class (genuine). Moreover, FAR_{ω} is introduced:

$$FAR_{\omega} = \omega.IAPMR + (1-\omega).FMR \tag{2.44}$$

which is weighted error rate for the two negative classes. Then, the operating threshold, $\tau_{\omega,\beta}$, is selected by minimizing the following formula on the development set:

$$\beta.FAR_{\omega} + (1-\beta).FNMR \tag{2.45}$$

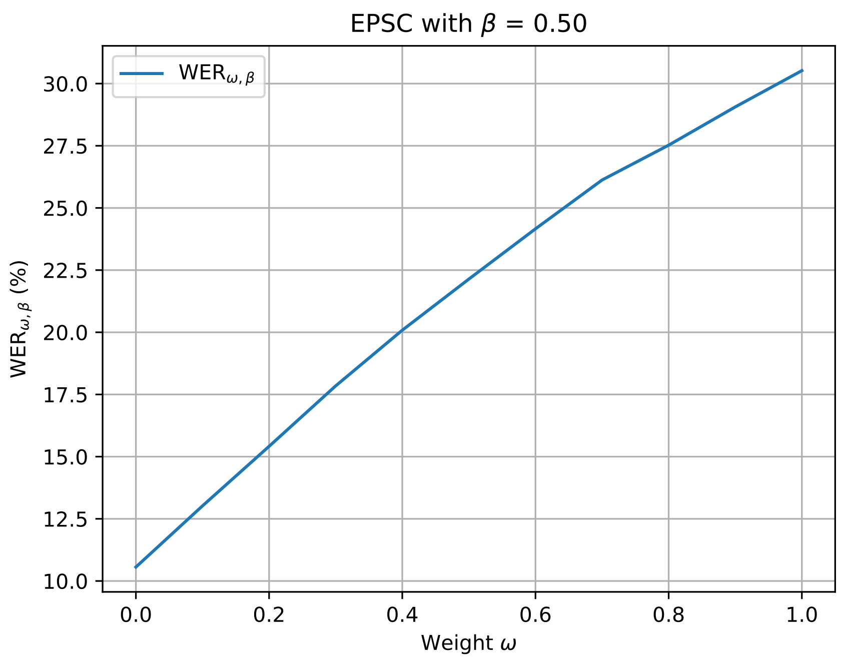
Once an optimal threshold is calculated for the given values of ω and β, different error rates can be computed on the evaluation set (Chingovska, Anjos, and Marcel, 2014; Chingovska et al., 2019). Chingovska, Anjos, and Marcel (2014) also introduce the following two new error rates:

$$WER_{\omega,\beta} = \beta.FAR_{\omega} + (1-\beta).FNMR \tag{2.46}$$

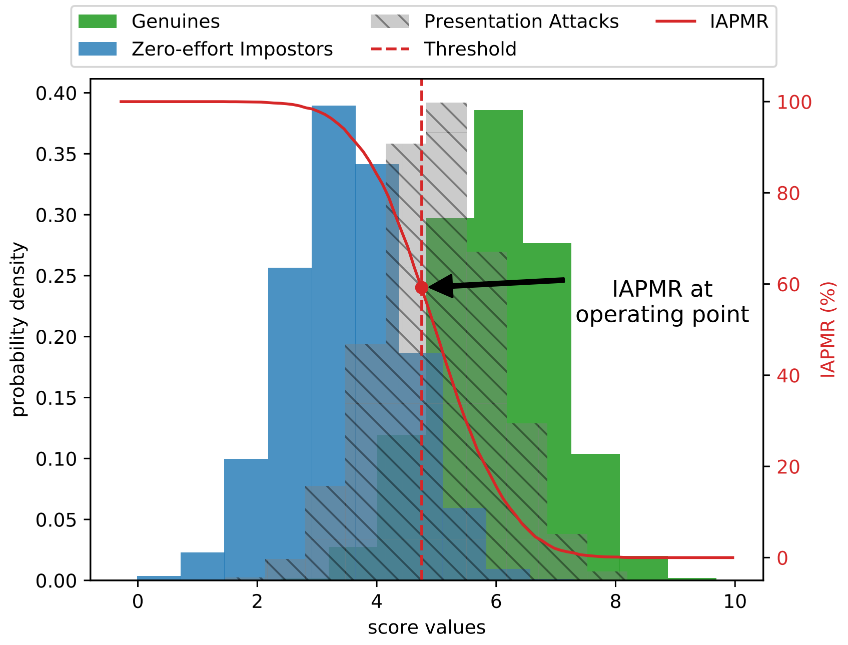
and when $\beta = 0.5$:

$$HTER_{\omega} = \frac{FAR_{\omega} + FNMR}{2} \tag{2.47}$$

Note that, in the EPS framework, similar to EPC, all thresholds must be calculated using the development set and all error rates must be calculated on the evaluation set. Again, similar to EPC, the **expected performance and spoofability curve (EPSC)** is also introduced where error rates are plotted against the cost parameter. In EPSC, because there are two cost parameters (ω and β), one parameter is fixed (usually at 0.5) and the curve is computed by varying the other parameter. Two common plots used in the evaluation of biometric verification systems are shown in figure 2.25.



(a) – An EPSC plot evaluating one biometric system. β (see equation (2.45)) is fixed to 0.5 and threshold for different ω values are computed on the development set. Then, error rates (here, $WER_{\omega,\beta}$) are computed on the evaluation set.



(b) – A histogram plot showing the score distribution of genuine, ZEI, and PA samples. The evolution of IAPMR as the threshold changes (different values on the x-axis) is also shown.

Figure 2.25 – Example plots of evaluation of biometric systems as a pseudo ternary classification problem.

2.7 Datasets

In this thesis, datasets for both face recognition and face PAD are used because both face recognition and face PAD systems are evaluated. In chapter 3, where the vulnerability of face recognition systems to presentation attacks is reported, the MOBIO dataset (McCool et al., 2012) (a face recognition dataset) is initially used to benchmark the performance face recognition systems in a licit scenario. Then, in chapter 3, Replay-Attack (Chingovska, Anjos, and Marcel, 2012), MSU-MFSD (Wen, H. Han, and Jain, 2015), and Replay-Mobile (Costa-Pazo et al., 2016) face PAD datasets are used to report the vulnerability of face recognition systems to presentation attacks.

In the rest of the thesis (chapters 4, 5 and 6), only the following face PAD datasets are used: Replay-Mobile (Costa-Pazo et al., 2016), OULU-NPU (Zinelabinde Boulkenafet et al., 2017), SWAN (which was collected during this thesis with colleagues (Ramachandra et al., 2019)), and WMCA (George et al., 2019). The Replay-Attack and MSU-MFSD face PAD datasets are skipped from the rest of the experiments because they are old datasets and the resolution of images are much lower compared to the newer face PAD datasets. The details of the datasets are given in the following.

2.7.1 MOBIO

The MOBIO dataset (McCool et al., 2012) was collected for bi-modal (voice and face) biometric verification experiments using mobile devices. Therefore, it contains only licit protocols. Biometric samples were collected in the year 2010, using Nokia N93i mobile phones, for 100 male subjects and 52 female subjects, spread over six cities (in five countries). The videos have been recorded at a resolution of (320 × 240) pixels. Experimental results reported in this thesis have been produced using face-images from only the male subjects.

2.7.2 Replay-Attack

The Replay-Attack (Chingovska, Anjos, and Marcel, 2012) face PAD dataset, published in 2012, consists of 1,300 videos (each, at least 9 seconds long) from 50 subjects. The videos have been recorded by using a 13″ Macbook at 320 × 240 pixel resolution, under two different lighting conditions. Three kinds of replay attacks are simulated in this dataset, namely, printed photo attacks, still face-images displayed on an electronic device, and replayed digital-videos. Three presentation attack instruments (PAIs) have been used to construct the attacks: photo-quality paper (for the printed photo attacks), iPhone 3Gs, and iPad (first generation). The two electronic devices have been used for both, photo- as well as video-attacks. Videos for each kind of attack have been captured under two conditions, one where the capturing device is hand-held, and the other where the capturing device rests on a stationary support.

2.7.3 MSU-MFSD

The public version of the MSU-MFSD dataset (Wen, H. Han, and Jain, 2015) includes real-access and attack videos for 35 subjects. This dataset was published in 2015. Real-access videos (~ 12 sec. long) have been captured using two devices: a 13″ MacBook Air (using its built-in camera), and a Google Nexus 5 (Android 4.4.2) phone. Videos captured using the laptop camera have a resolution of 640 × 480 pixels, and those captured using the Android camera have a resolution of 720 × 480 pixels. The dataset also includes PA videos representing printed photo attacks, mobile video replay-attacks where video captured on an iPhone 5s is played back on an iPhone 5s, and high-definition (HD) video-replays (captured on a Canon 550D SLR, and played back on an iPad Air).

2.7.4 Replay-Mobile

The Replay-Mobile dataset (Costa-Pazo et al., 2016) contains short (~ 10 sec. long) HD (720 × 1280) resolution videos corresponding to 40 identities, recorded using two mobile devices: an *iPad Mini 2* tablet and a *LG-G4* smartphone. The videos have been collected under six different lighting conditions, involving artificial as well as natural illumination. PAs represented in this dataset have been constructed using two PAIs: matte-paper for print-attacks, and matte-screen monitor for digital-replay attacks. For each PAI, two kinds of attacks have been recorded: one where the user holds the recording device in hand, and the second where the recording device is stably supported on a stand. Thus, four kinds of attacks are represented in the dataset. The *grandtest* protocol of the dataset was used in the experiments.

2.7.5 OULU-NPU

Table 2.12 – Details of OULU-NPU protocols. The table is adapted from (Zinelabinde Boulkenafet et al., 2017).

Protocol	Subset	Session	Phones	Users	Attacks created using
Protocol I	Train	Session 1,2	6 Phones	1-20	Printer 1,2; Display 1,2
	Dev	Session 1,2	6 Phones	21-35	Printer 1,2; Display 1,2
	Eval	Session 3	6 Phones	36-55	Printer 1,2; Display 1,2
Protocol II	Train	Session 1,2,3	6 Phones	1-20	Printer 1; Display 1
	Dev	Session 1,2,3	6 Phones	21-35	Printer 1; Display 1
	Eval	Session 1,2,3	6 Phones	36-55	Printer 2; Display 2
Protocol III	Train	Session 1,2,3	5 Phones	1-20	Printer 1,2; Display 1,2
	Dev	Session 1,2,3	5 Phone	21-35	Printer 1,2; Display 1,2
	Eval	Session 1,2,3	1 Phone	36-55	Printer 1,2; Display 1,2
Protocol VI	Train	Session 1,2	5 Phones	1-20	Printer 1; Display 1
	Dev	Session 1,2	5 Phones	21-35	Printer 1; Display 1
	Eval	Session 3	1 Phone	36-55	Printer 2; Display 2

The OULU-NPU dataset (Zinelabinde Boulkenafet et al., 2017) is a dataset published in 2017. It dataset contains short (~ 5 sec. long) HD (1080×1920) resolution videos corresponding to 55 identities, recorded using six mobile devices (biometric sensors): *Samsung Galaxy S6 edge, HTC Desire EYE, MEIZU X5, ASUS Zenfone Selfie, Sony XPERIA C5 Ultra Dual* and *OPPO N3*. The recordings are done in three different sessions with variation in illumination and background. There are two types of PAs: print and video-replay. Each type was created using two different presentation attack instrument (PAI): two printers for print attacks and two displays for video-replay attacks.

The dataset comes with 4 different protocols. Each protocol changes one or several criteria between *training*, *development*, and *evaluation* sets to evaluate the performance of a PAD system when mismatches are present between sets. In all protocols, identities do not overlap between sets. Protocol I uses both sessions 1 and 2 in both *Train* and *Development* sets while session 3 is used in the *Evaluation* set. The sessions differ in illumination and background scenes. Protocol II uses different PAIs between sets. It uses printer 1 and display 1 in both *Train* and *Development* sets while printer 2 and display 2 is used in the *Evaluation* set. Protocol III uses different biometric sensors between sets. It uses the data from 5 of the phones in both *Train* and *Development* sets while the data from 1 phone is only used in the *Evaluation* set. There are 6 folds by selecting 1 phone out of 6 phones for the *Evaluation* set in protocol III. Finally, protocol IV contains the variations of all three former protocols. It uses sessions 1 and 2, printer 1, display 1, and 5 phones in both *Train* and *Development* sets while session 3, printer 2, display 2, and 1 phone are used in the *Evaluation* set. Similar to protocol III, protocol IV also comes with 6 folds. Details of the protocols are shown in table 2.12.

Unfortunately, protocols of the OULU-NPU dataset are not designed in a way to support purely data-driven approaches to face PAD. For example, I argue that to train a PAD system that can generalize against different illumination and backgrounds, i.e. sessions, at least 4 sessions are needed in the dataset; 2 sessions to be used in the *Train* set so that the system is exposed to at least two different sessions in the training, 1 session to be used in the *Development* set to make sure the PAD system does not overfit on the training sessions, and 1 session to evaluate the PAD method on unseen sessions. However, it is not possible to implement such a protocol using the OULU-NPU dataset because there are only 3 sessions in the dataset. This is also true for other protocols of the dataset. In all protocols the *Development* set data is has no differences (except for identities) to the *Train* set which makes them unsuitable for evaluating data-driven PAD methods. This will be discussed further in section 4.2.

2.7.6 SWAN

The SWAN dataset is a smartphone multimodal biometric authentication dataset which contains multimodal recordings of 150 identities from multiple locations: Switzerland, Norway, France, and India. The recordings contain face images and videos, audio-visual talking faces videos, and periocular images and videos. Each identity is recording in 6 different sessions where the interval between sessions ranges from 1 day to 3 weeks reflecting real-world conditions. The capture environment includes both indoor and outdoor scenarios in assisted/supervised and unsupervised capture settings. The data capture includes both self and assisted capture processes replicating the real-life applications such as banking transactions. Further, presentation attacks for each modality is created. In case of face, there is one print attack and two replay video attacks (using iPhone 6 and iPad PRO as PAI) reflecting low and high quality replay attacks, respectively. Being a new smartphone multimodal biometric dataset with presentation attack samples, it allows one to develop and benchmark both verification and Presentation Attack Detection (PAD) algorithms.

2.7.7 WMCA

The *wide multi-channel presentation attack* (WMCA) dataset by (George et al., 2019) is a face PAD dataset with a wide variety of 2D and 3D presentation attacks, specifically, 2D print and replay attacks, mannequins, paper masks, silicone masks, rigid masks, transparent masks, and non-medical eyeglasses. The 10 second face image videos are recorded using multiple devices which results in the final multi-channel videos made of color, depth, near-infrared, and thermal. There are bona fide and presentation attack recordings of 72 identities. The bona fide recordings were done in 7 sessions during 5 months with different background and illumination in each session. Only 2D print and replay attacks and only RGB images of the WMCA dataset were used in this study to be able to compare to other datasets.

3 A Study of The Robustness of Face Recognition to Presentation Attacks

This chapter is adapted from the post-print version of the following publication:

Amir Mohammadi, Sushil Bhattacharjee, and Sébastien Marcel (2017). “Deeply Vulnerable: A Study of the Robustness of Face Recognition to Presentation Attacks”. In: *IET Biometrics* 7.1, pp. 15–26

Most high-accuracy face recognition (FR) systems today rely on deep-learning methods. Indeed, deep-learning based FR systems are already being deployed in commercial face-verification applications (Schroff, Kalenichenko, and Philbin, 2015). In this context their vulnerability to presentation attacks (PA) becomes an important factor in the trustworthiness of the entire face-verification process. Several studies (Taigman et al., 2014; Parkhi, Vedaldi, and Zisserman, 2015; Schroff, Kalenichenko, and Philbin, 2015; Y. Sun et al., 2015) have explored the capacity of deep-learning based FR-systems to handle variations in pose, illumination and scale, that is, challenges related to variability of face-biometric samples of a single client. Karahan et al. (2016) studied the fragility of CNN-based FR systems when confronted with image degradations such as blurring, occlusion, compression-artifacts, and color-distortion. Robustness to such degradations is necessary for FR systems operating in relatively unconstrained environments. For face-verification systems functioning under controlled conditions, the vulnerability to PAs is a much more significant concern.

The European research project TABULA RASA (*Trusted Biometrics under Spoofing Attacks (TABULA RASA)* 2016) indicated a positive correlation between the efficacy of FR systems and their vulnerability to PAs. In other words, FR methods with higher face-verification accuracy tended to be more vulnerable to PAs. Given that FR systems, especially the popular CNN-based FR methods, have not been explicitly trained for presentation attack detection (PAD), one would intuitively expect such systems to be vulnerable to PAs. The vulnerability of deep learning based FR systems to PAs has not been studied in detail.

In this chapter, we present a large-scale empirical study of the vulnerability to PAs of five recent FR systems, including 3 recent CNN-based methods. Specifically, we compare the vulnerability of the following systems:

- three CNN-based FR systems, namely:
  - VGG-Face (Parkhi, Vedaldi, and Zisserman, 2015),
  - LightCNN (X. Wu et al., 2015), and
  - FaceNet (Sandberg, 2017)
- Intersession Variability (ISV) modeling (Wallace et al., 2011), and
- ROC-SDK from Rank-One Computing (SDK version 1.9)[1].

These systems were detailed in section 2.3 on page 27.

The main contribution of this chapter is empirical evidence to support the claim that the CNN-based FR method is extremely vulnerable to PAs. Our experiments, using three PAD datasets, show that other highly rated FR methods are also very vulnerable to PAs.

3.1 Related Work

There have been very few studies specifically exploring the vulnerability of FR systems to different kinds of attacks. Duc and Minh (2009) have investigated the vulnerability of FR based login on computers (specifically Lenovo, Toshiba and Asus products) to PAs. Kose and Dugelay (2013) present a study of the vulnerability of FR methods to 3D-mask attacks. They evaluate two different FR methods – one designed specifically for FR using 3D data (Erdogmus and Dugelay, 2012), and, a generic approach to FR using local binary patterns (LBP) based face-image comparison. Hadid (2014) discusses the vulnerability of the parts-based GMM FR method to PAs. In a recently published work, Scherhag et al. (2017) have studied the vulnerability of FR methods to morphed-face attacks. Here attacks are constructed by morphing face images from two enrolled identities, and the challenge is to see if the two enrolled identities can both be successfully spoofed using the same morphed image. Both FR methods tested in this work – VeriFace SDK[2] (a commercial off-the shelf (COTS) FR product from Neurotechnology), and OpenFace (Amos, Ludwiczuk, and Satyanarayanan, 2016) (an open-source, pre-trained DNN based FR system) – are shown to be highly susceptible to such attacks (Scherhag et al., 2017).

Although deep-learning based FR systems have attracted considerable attention, both academic and commercial, as far as we are aware, no previous study has investigated the vulnerability of CNN-based FR systems using a variety of PAs. Most of the studies cited above have been performed on FR methods that rely on hand-crafted features. The study by Scherhag et al. (2017) that has included a DNN based FR system (OpenFace) was done in the restricted

[1]Company website: www.rankone.io

[2]Website: www.neurotechnology.com/verilook.html

context of a single class of attacks based on morphed-images. Such attacks are expected to have a very narrow range of application.

In this work, we focus on the vulnerability of CNN based FR methods to PAs. This study includes three publicly available pre-trained CNN-FR models. Our experiments demonstrate that all these FR methods are consistently highly vulnerable to several classes of PAs. For comparison, we have also included the ISV modeling method, which relies on hand-crafted features, and ROC-SDK – a COTS FR product – in this study. We use four publicly available FR and PAD datasets to estimate the vulnerability of the five FR methods to a variety of PAs. Our study has been motivated by the hypothesis that although CNN-FR systems outperform FR methods based on hand-crafted features, they are also more vulnerable to PAs. This assertion makes intuitive sense to researchers in face-biometrics, given that the FR methods have not been explicitly trained to detect PAs. However, this is a large-scale study to empirically quantify the vulnerability of these FR methods.

3.2 Experiment Datasets and Protocols

3.2.1 Datasets

We have used four publicly available datasets, namely, MOBIO (McCool et al., 2012), Replay-Attack (Chingovska, Anjos, and Marcel, 2012), MSU-MFSD (Wen, H. Han, and Jain, 2015), and Replay-Mobile (Costa-Pazo et al., 2016). These datasets were detailed in section 2.7 on page 59.

The various PAs represented in the three PAD datasets are summarized in table 3.1. In this table, **PA I** refers to printed photo attacks, **PA II** refers to low-quality replay attacks, and **PA III** denotes high-quality replay attacks. Figure 3.1 shows some example PAs from each PAD dataset. The first column shows examples of *bona fide* presentations. The remaining three columns show examples of selected PAs represented in the corresponding dataset.

3.2.2 Protocols

The MOBIO dataset, which is an FR dataset, comes with only a licit protocol, whereas the PAD datasets include both licit and spoof protocols (see section 2.6.2 on page 54 for definition of licit and spoof protocols). For our experiments with the MSU-MFSD, we have constructed new licit and spoof protocols to analyze the vulnerability of the FR methods to PAs. The protocols in the various datasets are described in this section.

The set of clients in the licit protocol of each dataset is divided into three mutually exclusive *groups*: *training, development* and *evaluation.* The training group is reserved for producing the background models, when necessary. In our experiments, the UBM required for the ISV modeling method has been trained using the training group in the licit protocol of the MOBIO dataset. The training groups in the licit protocols of the PAD datasets are ignored here. The development and evaluation groups, each consist of two *sets* of samples: an *enrollment* set

Table 3.1 – Different combinations of PA in each dataset. **PA I** refers to printed photo attacks in the dataset, **PA II** refers to low-quality video replay attacks, and **PA III** corresponds to high-quality video replay attacks. The Replay-Mobile dataset contains only two PA types: **PA I** and **PA III**. Low-quality video replay attacks have not been included in this dataset.

Dataset		**PA I**	**PA II**	**PA III**
Replay-Attack	**Capture Device**	12.1 mega-pixel Canon PowerShot SX150 IS camera	3.1 mega-pixel iPhone 3GS camera	3.1 mega-pixel iPhone 3GS camera
	PAI	Triumph-Adler DCC 2520 color laser printer	iPhone 3Gs (320 × 480 pixels) (165 ppi pixel density)	iPad (768 × 1024 pixels) (132 ppi pixel density)
	Capture Conditions	2 lighting conditions ('normal' (bright) and 'adverse' (in the shade, but not dark)) Hand-held or fixed		
	Biometric sensor	Macbook (320 × 240)		
MSU-MFSD	**Capture Device**	Canon 550D camera	8 mega-pixel iPhone 5S camera	Canon 550D camera
	PAI	HP Color Laserjet CP6015xh printer	iPhone 5s (640 × 1136 pixels) (326 ppi pixel density)	iPad Air (1536 × 2048 pixels) (264 ppi pixel density)
	Capture Conditions	Indoors, with artificial lighting		
	Biometric sensor	Macbook Air (640 × 480), Nexus 5 (720 × 480)		
Replay-Mobile	**Capture Device**	18 mega-pixel Nikon Coolpix P520		LG-G4 (720 × 1280)
	PAI	Konica Minolta ineo+ 224e color laser printer		Philips 227ELH matte monitor (1920 × 1080)
	Capture Conditions	6 lighting conditions (including artificial lights and natural light) Hand-held or fixed		
	Biometric sensor	iPad Mini 2 (720 × 1280), LG-G4 (720 × 1280)		

and a *probe* set. All clients assigned to the group are represented in both sets. That is, for each client in the development group, a certain number of enrollment samples as well as probe samples are available (and likewise, for the evaluation group).

Spoof protocols of the various PAD datasets also consist of three non-overlapping groups: training, development, and evaluation. The training group is intended to be used for training a PAD method. The development group can be used for tuning the parameters of the PAD method being tested, and the final performance metrics are reported for the evaluation group. In the vulnerability analysis experiments using a given PAD dataset, attack-videos in the evaluation group of the corresponding PAD protocol are used to test the vulnerability of the FR system in question.

The three groups in the male protocol of the MOBIO dataset are constructed as follows:

- training: 7881 samples images, from 37 subjects,
- development: 2760 samples, from 24 subjects, and
- evaluation: 4370 samples, from 38 subjects.

As mentioned before, samples for the MOBIO dataset were collected at six different sites. Each group contains data exclusively from two sites.

The Replay-Attack dataset comes with a licit protocol consisting of 100 videos (one video per subject, per illumination condition), and several PAD protocols. The licit protocol is partitioned thus:

- training: two videos each, from 15 subjects,

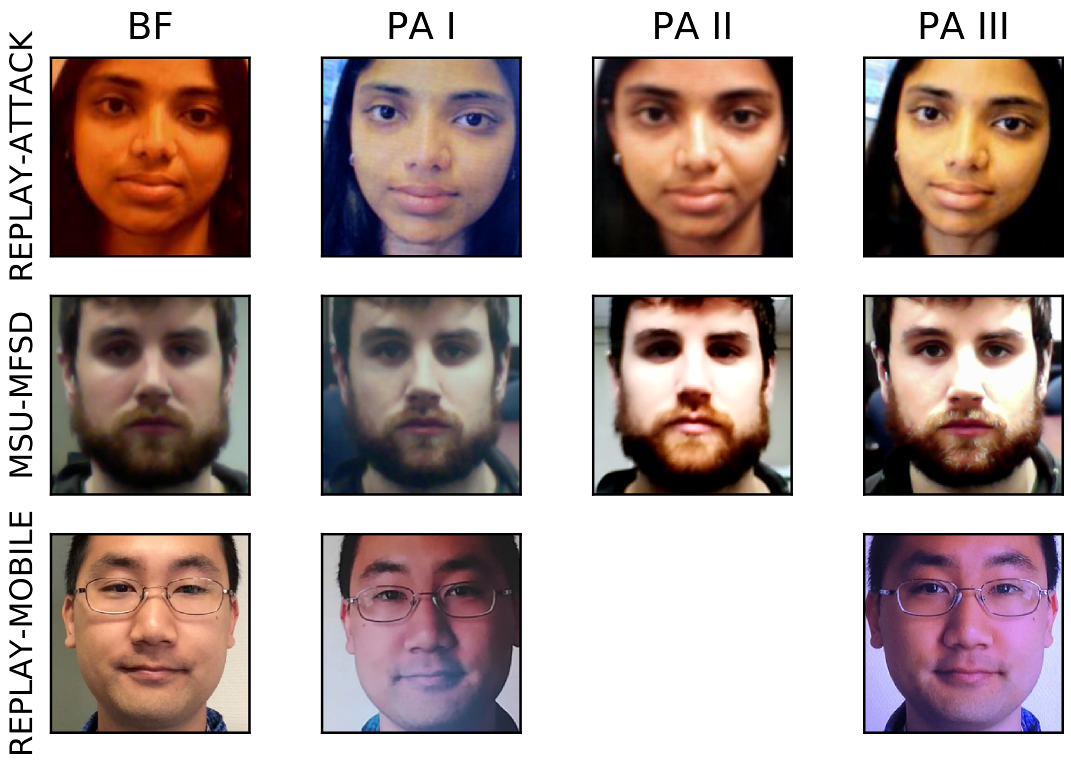


Figure 3.1 – Examples of attacks represented in the various PAD datasets. The first column shows a *bona fide* example, and the remaining columns show different types of PAs present in the dataset for the same identity. The various PAs are described in table 3.1.

- development: two videos each, from 15 subjects, and
- evaluation: two videos each, from 20 subjects.

The dataset also provides several PAD protocols (Chingovska, Anjos, and Marcel, 2012). In our experiments we have considered the *grandtest* protocol, which includes all PAs. This protocol consists of 200 *bona fide* presentations, and 1,000 PAs (for a total of 1,200 videos).

The public version of MSU-MFSD (Wen, H. Han, and Jain, 2015) offers 70 *bona fide* presentation videos and 280 attack videos. As mentioned before, no licit protocol has been published for this dataset. For our experiments, therefore, we have devised a new licit protocol using some frames from the *bona fide* presentation videos. For each subject, the dataset contains two *bona fide* presentation videos, one captured using a Macbook Air (labeled ”laptop”) and the other using a smart-phone (labeled ”android”). To construct the licit protocol for this dataset, we have used 10 frames from each *bona fide* video, sampled evenly, between the first and 180^{th} frame. Frames from the ”laptop” videos form the enrollment set, and those from the ”android” videos form the probe set.

The licit protocol of the Replay-Mobile dataset consists of 160 videos (four videos for each of

40 subjects). They are grouped as follows: 48 videos (12 subjects) for training, 64 videos (16 subjects) for development, and the remaining 48 videos for the test group. The PAD protocol of the dataset consists of 1,030 videos (390 *bona fide* presentations and 640 PAs of different kinds).

For ease of reference, the number of clients and samples used in the licit and spoof protocols of the various datasets is summarized in table 3.2.

Table 3.2 – Composition of the licit and spoof protocols of the four datasets used in our experiments. All protocols consist of three mutually exclusive groups of clients: training, development, and evaluation. For each dataset, the number of clients assigned to each group is listed here. The number in parentheses shows the total number of samples for the corresponding dataset and group.

Dataset	**Licit protocol**			**Spoof protocol**		
	training	**development**	**evaluation**	**training**	**development**	**evaluation**
MOBIO	37 (7881)	24 (2760)	38 (4370)			
Replay-Attack	15 (90)	15 (90)	20 (120)	15 (300)	15 (300)	20 (400)
MSU-MFSD	10 (200)	10 (200)	15 (300)	10 (600)	10 (600)	15 (900)
Replay-Mobile	12 (1680)	16 (2240)	12 (1580)	12 (1920)	16 (2560)	12 (1920)

3.3 Experiments

The procedure used here to evaluate the vulnerability of a FR system is as follows. First the face-verification performance of the system is evaluated using only the MOBIO dataset (*i.e.,*, in the licit scenario). The hyper-parameters of the FR system are tuned to achieve optimal discrimination between genuine presentations and ZEI presentations. The vulnerability of the FR system to PAs is then evaluated, for the different PAD datasets, using the same hyper-parameter settings. In this section we first describe the methodology adopted for the experiments. Then, we present the results of face-recognition and vulnerability analysis for the five selected FR systems.

3.3.1 Methodology

The FR and vulnerability-analysis experiments have been performed using the Python based Bob[3] signal-processing and machine-learning toolkit (Anjos et al., 2012; Anjos et al., 2017). Besides a wide selection of machine-learning and signal processing tools relevant for biometrics experiments, the Bob toolkit also includes interfaces for accessing the various datasets and associated protocols. All the FR systems studied here have been tested within the same software framework.

The inputs for the LightCNN, ISV modeling and ROC-SDK FR methods are gray-scale images, and the input to the VGG-Face and FaceNet FR methods is a color (RGB) image. The ROC-SDK

[3]Website: www.idiap.ch/software/bob/

takes the entire image (one frame of video) as input. All other FR methods expect the input-image to be cropped appropriately, so that it contains only the face-region, with as little of the background as possible (see section 2.3.1 on page 28). The specific input image dimensions required by the various FR systems, are listed in table 3.3. The table also summarizes the hyper-parameters and the scoring method used in each FR system. For the ISV modeling method, we have used the hyper-parameter values recommended by Günther, El Shafey, and Marcel (2016).

The ROC-SDK as well as the three CNN-FR methods (VGG-Face, LightCNN, and FaceNet) come pre-trained, and can be used out of the box. The UBM and the session-variability matrix ($\boldsymbol{U}$ in equation (2.20)) for the ISV modeling method are trained using training set in the licit male-protocol of the MOBIO dataset.

Table 3.3 – Summary of the parameters of the various FR systems used in this study. All methods, apart from ROC-SDK, expect input-images of fixed dimensions, where the image has been cropped to the face-region appropriately.

FR Method	Input	Features	Hyper-parameters	Comparison Metric
VGG-Face	224×224 color image	Output of 'fc7' layer length 4096	None (pre-trained CNN)	Cosine distance measure
LightCNN	128×128 gray image	Output of 'eltwise_fc1' layer length 256	None (pre-trained CNN)	Cosine distance measure
FaceNet Model: 20170512-110547	160×160 color image	Output of 'embeddings:0' layer length 128	None (pre-trained CNN)	Cosine distance measure
ISV	80×64 gray image	Session-variability-compensated supervector, length $\sim 45K$	#GMMs: 512; Dim. of session-variability subspace, U: 160	Log likelihood ratio
ROC-SDK (v1.9)	arbitrary size gray image	Undisclosed, proprietary size ~ 140 bytes	Using FRONTAL, FR, PARTIAL, and ROLL filters	Undisclosed, proprietary

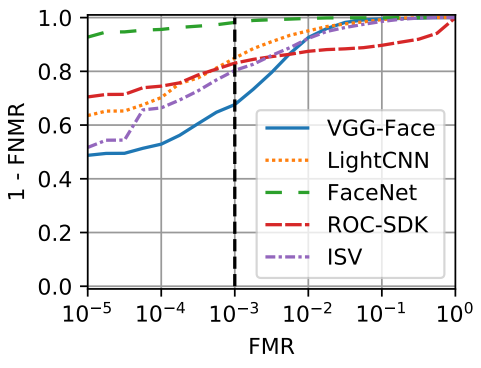
For experiments with each dataset, the hyper-parameters of each FR method are tuned using data in the development group of the licit protocol. The score-threshold may be chosen arbitrarily, but usually some heuristics are applied when choosing the threshold. Two common approaches for selecting the score-threshold are:

- Select the score-threshold, $\mathcal{T}_{EER}$, corresponding to the equal error rate (EER). That is, $\mathcal{T}_{EER}$ is the threshold for which $FMR \approx FNMR$.
- Select the score-threshold, $\mathcal{T}_{\theta}$, that corresponds to a specific FMR = θ (*e.g.*, $\mathcal{T}_{0.1}$ denotes the score-threshold leading to a FMR of 0.1% over the development group).

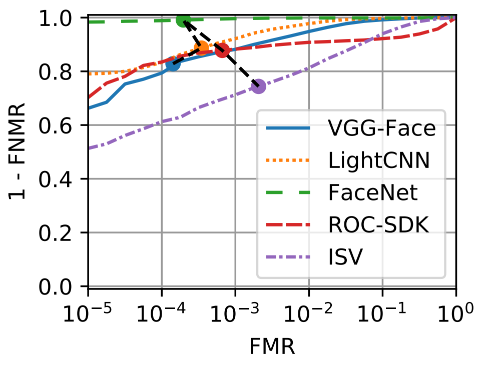
For a given score-threshold obtained on the development group, the FMR, FNMR, and HTER can be reported on the evaluation group.

3.3.2 Face Verification Performance

First, we establish a baseline face-verification performance for each FR system, based on the licit protocol of the MOBIO dataset. The face-verification results of the various FR systems are presented in two ways: using receiver operating characteristics (ROC) curves and using



(a) – ROC curves for the development group.



(b) – ROC curves for the evaluation group.

Figure 3.2 – Performances of FR systems in licit scenario of the MOBIO dataset. (a) ROC curves for the development group of the licit protocol of MOBIO. The vertical dashed line marks the point where FMR is 0.1% for every method (corresponding to the score-threshold $\mathcal{T}_{0.1}$). (b) ROC curves for the evaluation group of the licit protocol of MOBIO. The colored circular markers (connected by a dashed line) indicate the FMR and FNMR of each FR method, for the $\mathcal{T}_{0.1}$ score-threshold.

expected performance curves (EPC) (S. Bengio, Mariéthoz, and Keller, 2005) (see section 2.6 on page 48).

The ROC curves shown in figure 3.2 illustrate the face-verification performances of the various FR systems. They have been evaluated using the licit male-protocol of the MOBIO dataset. Figure 3.2(a) and (b) show the ROC plots for the development group and evaluation group, respectively. A score-threshold, $\mathcal{T}_{0.1}$, is selected so as to achieve a FMR of 0.1% over the development group. This is indicated in figure 3.2(a) by a dashed black vertical line. In figure 3.2(b) the circular markers in the various colors (connected by a dashed trace line) indicate the FMR of the corresponding FR methods for the score-threshold $\mathcal{T}_{0.1}$. To compare the performances of the five FR methods on the evaluation group at score-threshold $\mathcal{T}_{0.1}$, we should consider the distances of these five points from the top left corner of the plot (where $(FMR, FNMR) = (0,0)$, representing the ideal outcome). This way of comparing different FR systems follows the method used by NIST (Grother, Ngan, and Hanaoka, 2017). It is clear from the ROC plots that the FaceNet CNN significantly outperforms the other FR methods in this study. Indeed, it is noteworthy that the verification-accuracy of the FaceNet CNN remains extremely high even for very small values of FMR.

Expected performance curves (EPC) (S. Bengio, Mariéthoz, and Keller, 2005) provide an unbiased comparison of FR systems. Figure 3.3 shows the EPC plots of the five FR systems. These curves have been computed using the licit protocol of the MOBIO dataset, and show how the performance of each FR method evolves as a function of the trade-off between FMR and FNMR. This trade-off is controlled by the parameter $0 \leq \alpha \leq 1$ as follows:

$$WER = \alpha \times (FMR) + (1 - \alpha) \times (FNMR) \ , \tag{3.1}$$

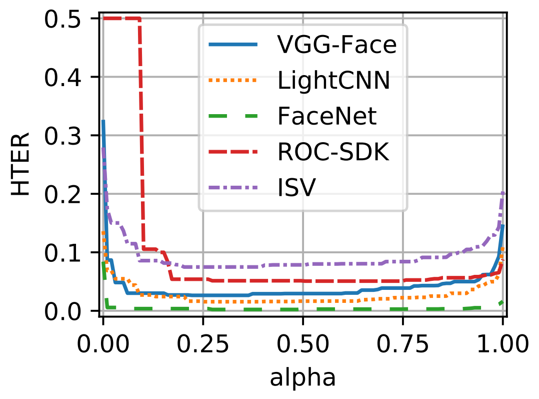


Figure 3.3 – Comparison of FR systems in licit scenario. The plot shows EPC curves for the five FR systems. The licit protocol of the MOBIO dataset was used to generate these curves. This plot confirms the conclusion drawn from figure 3.2, that the FaceNet CNN significantly outperforms the other studied FR methods.

where WER (weighted error rate) is the weighted combination of FMR and FNMR. As always, the WER is computed from the classification results of the development group. Any particular value of WER corresponds to specific values of α, FMR and FNMR. Therefore, to achieve a specific WER, for a given value of α, the score-threshold should be chosen appropriately. Specifically, when constructing EPC, we consider a discrete set of values for α. For each value of α, the score-threshold that minimizes the WER for the development group is chosen. The selected score-threshold is then used to compute the HTER over the evaluation group, shown in the EPC plots (figure 3.3). Thus, the α values (on the horizontal axis) and the HTER values of the evaluation group (on the vertical axis of the plot) are related via the corresponding score-threshold computed from the development group. Figure 3.3 confirms that the three CNN based FR methods perform better than the other two methods (ISV modeling and ROC-SDK). In particular, the figure also shows that the FaceNet CNN promises significantly better performance than all the other FR methods, including the popular VGG-Face network.

3.3.3 Discussion of Face Verification Results

Comparing the two plots in figure 3.2, we note that (for the MOBIO male dataset) the performance of the VGG-Face CNN is not consistent over the two groups. At the operating point corresponding to score-threshold $\mathcal{T}_{0.1}$, on the development group, the ROC-SDK and the ISV modeling method, both outperform the VGG-Face CNN. For the evaluation group, however, the VGG-Face network performs much better. Figure 3.2(b) shows that the FMR for the VGG-Face CNN is at least one order of magnitude smaller than the FMR for both the ROC-SDK and the ISV modeling method. By contrast, the LightCNN and FaceNet networks consistently perform better than the other FR methods on both groups.

The variable α in equation (3.1) is a cost-variable that can be adjusted depending the use-case.

If minimizing FMR (Type I error) is critical, then high values of α are used to determine the score-threshold from the development group. In the less likely scenario where minimizing FNMR (Type II error) is of utmost importance, very low values of α should be used. When it is important to optimize both types of errors, α is set to 0.5, which corresponds to determining the score-threshold $\mathcal{T}_{EER}$.

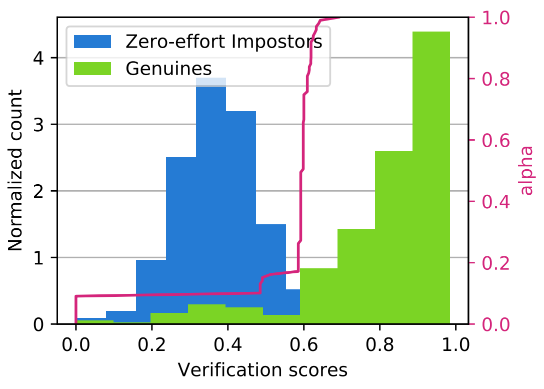
The EPC plots also indicate that the FaceNet CNN shows near perfect FR performance over almost the entire range of α values.

Note the initial horizontal segment of the EPC curve for ROC-SDK, indicating a very high HTER for values of $\alpha \leq 0.1$. This results from the peculiarity of the score-distribution produced by the ROC-SDK method. To explain this behaviour, let us look at the score-distributions shown in figure 3.4. The score-distributions produced by the ROC-SDK for the two classes (genuine presentations in green, and ZEI-presentations in blue) are shown in figure 3.4(a). The score-threshold for optimal separation between the two classes increases (along the horizontal axis) with α. The red line shows this evolution of the score-threshold. Note that the histograms shown in the plot represent score-distributions of the evaluation group, whereas the red line shows the evolution of the score-threshold as computed over the development group. Figure 3.4(b) shows the score-distributions and score-threshold evolution produced by the VGG-Face CNN.

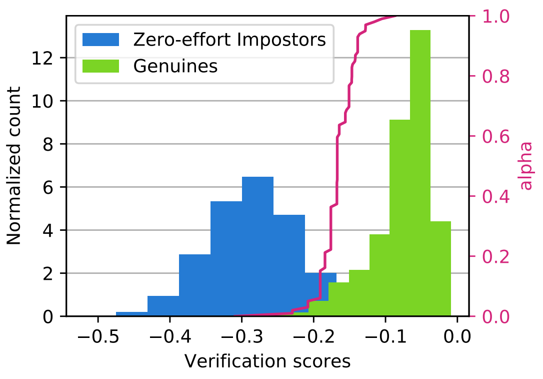
We can see in figure 3.4(a) that for some genuine presentations the ROC-SDK produces very low scores, which are similar to scores of ZEI presentations. Therefore, for low values of α, where the goal is to minimize FNMR even at the cost of high FMR, the selected score-threshold is so low that almost every presentation is accepted. This leads to the near-50% HTER. We do not see this behaviour in the EPC curves of the other FR methods because these methods do not produce abnormally low scores for genuine presentations. Figure 3.4(b) shows that the two score-distributions produced by the VGG-Face network do not have significant overlap, and therefore, even for low values of α, the selected score-threshold is not very low (relative to the score-distribution of the ZEI class). Therefore, we see a gradual change at the beginning of the EPC curve for the VGG-Face method, and not a sudden drop as seen for the ROC-SDK method. Of course, this analysis applies only to the MOBIO male dataset. For other datasets, the score-distributions produced by the ROC-SDK for the two classes need not have significant overlap.

3.3.4 Vulnerability Analysis

The trained FR systems are next evaluated with respect to their vulnerability to PAs, using the PAD datasets: Replay-Attack, MSU-MFSD, Replay-Mobile, and a combined PAD dataset, created by merging the three PAD datasets. In the combined dataset, the licit protocol consists of the licit protocols of the individual PAD datasets, and similarly, the spoof-protocol includes the spoof-protocols of the three PAD datasets. In each protocol of the combined dataset, the development group is composed of the development groups of the three constituent



(a) – Score-distribution produced by the ROC-SDK.



(b) – Score-distribution produced the VGG-Face CNN.

Figure 3.4 – Score-distributions of two FR systems in the licit scenario of MOBIO dataset. (a) Histograms of scores produced by the ROC-SDK method. (b) Score-histograms for the VGG-Face network. In each plot, the green histogram represents the distribution of scores for genuine presentations, and the blue histogram shows the scores for the ZEI presentations. These histograms are based on scores computed for the evaluation group. In each plot, the red line shows the evolution (from left to right) of the score-threshold computed from the development group, as a function of α (right vertical axis).

PAD datasets, and likewise for the evaluation group. First, the development group of the licit protocol in each PAD dataset is used to determine a score-threshold for classifying genuine- and ZEI-presentations. Presentations in the evaluation group of the licit protocol are then classified using this score-threshold. These classification results are reported using the FMR and FNMR metrics. The presentations in the spoof protocol of the dataset are also classified using the same score-threshold, and the results are used to compute the IAPMR.

Vulnerability-analysis results for the five FR systems are summarized in table 3.4. For each FR system, the table shows the results for the three PAD datasets individually, as well as the combined PAD dataset. The table shows both the face-verification accuracy as well as the vulnerability of each FR system. Values of the different metrics are reported for two score-thresholds – $\mathcal{T}_{EER}$, and $\mathcal{T}_{0.1}$ – determined using the development group. The values of the performance-metrics shown in the table have been computed over the evaluation groups of the various datasets. On the combined dataset the LightCNN shows the same face-verification performance as the VGG-Face system. However, the general conclusion drawn from the HTER values in the table is that the FaceNet CNN achieves the best face-verification performance for all three PAD datasets, as well as for the combined dataset.

In table 3.4, the IAPMR values for the all three CNN-FR systems are consistently above 90% for all PAD datasets (using both score-thresholds). In fact, when we consider the lower-bounds of the respective confidence-intervals, in a majority of experiments, the CNN-FR systems show vulnerabilities higher than 95%. These experiments clearly demonstrate that the CNN based FR systems are highly vulnerable to PAs. Comparing the IAPMR values in the table,

Table 3.4 – FR accuracy and vulnerability of each FR system are shown for four datasets (the three PAD datasets, as well as the combined dataset). The FR accuracies of five FR methods are reported in terms of FMR and FNMR. For convenience, the HTER is also reported for each experiment. The table also shows the vulnerability of each FR method to presentation attacks (IAPMR). The values in the table have been computed on the evaluation group, based on two different score-thresholds: $\mathcal{T}_{EER}$ (corresponding to the EER on the development group), and $\mathcal{T}_{0.1}$ (leading to a FMR of 0.1% on the development group). 95% confidence intervals are shown for the IAPMR values in brackets. The highest IAPMR and the lowest HTER values for each dataset and score-threshold are highlighted in bold.

FR Method	**Dataset**	**Score-Threshold @ EER** ($\mathcal{T}_{EER}$)				Score-Threshold @ FMR = 0.1% ($\mathcal{T}_{0.1}$)			
		FMR	**FNMR**	**HTER**	**IAPMR**	**FMR**	**FNMR**	**HTER**	**IAPMR**
VGG-Face	Replay-Attack	0.0	0.0	**0.0**	**98.2** [96.4, 99.3]	0.6	0.0	0.3	**99.8** [98.6, 100]
	MSU-MFSD	0.0	4.0	2.0	92.4 [90.5, 94.1]	0.0	1.3	0.7	93.1 [91.3, 94.7]
	Replay-Mobile	0.1	2.5	1.3	95.4 [94.3, 96.3]	0.0	4.4	2.2	90.7 [89.3, 92.0]
	Combined	0.3	2.0	1.2	97.0 [96.4, 97.6]	0.0	3.8	1.9	92.7 [91.7, 93.5]
LightCNN	Replay-Attack	0.1	0.0	**0.0**	95.0 [92.4, 96.9]	0.1	0.0	**0.1**	98.0 [96.1, 99.1]
	MSU-MFSD	0.0	0.0	**0.0**	93.4 [91.6, 95.0]	0.4	0.0	0.2	**99.9** [99.4, 100]
	Replay-Mobile	0.9	1.5	1.2	**99.9** [99.6, 100]	0.3	2.1	1.2	**99.7** [99.3, 99.9]
	Combined	1.0	1.4	1.2	**99.8** [99.6, 99.9]	0.5	1.7	1.1	**99.6** [99.3, 99.8]
FaceNet	Replay-Attack	0.0	0.0	**0.0**	**99.5** [98.2, 99.9]	0.5	0.0	0.2	**99.5** [98.2, 99.9]
	MSU-MFSD	0.0	0.0	**0.0**	**99.0** [98.1, 99.5]	0.0	0.0	**0.0**	**100** [99.6, 100]
	Replay-Mobile	0.0	0.5	**0.3**	**99.9** [99.6, 100]	0.0	0.5	**0.3**	**99.8** [99.5, 100]
	Combined	0.1	0.5	**0.3**	**99.9** [99.7, 100]	0.0	0.5	**0.2**	**99.8** [99.6, 99.9]
ROC-SDK	Replay-Attack	2.4	1.2	1.8	89.5 [86.1, 92.3]	0.3	3.8	2.0	87.0 [83.3, 90.1]
	MSU-MFSD	14.5	10.7	12.6	83.3 [80.7, 85.7]	0.0	16.0	8.0	77.3 [74.5, 80.0]
	Replay-Mobile	3.9	3.6	3.8	91.5 [90.1, 92.7]	0.0	5.2	2.6	87.5 [86.0, 89.0]
	Combined	5.0	4.8	4.9	87.8 [86.6, 88.9]	0.1	6.7	3.4	84.5 [83.2, 85.7]
ISV	Replay-Attack	0.4	0.0	0.2	92.2 [89.2, 94.7]	0.1	1.2	0.7	82.0 [77.9, 85.6]
	MSU-MFSD	0.2	12.7	6.5	90.8 [88.7, 92.6]	0.9	6.7	3.8	95.0 [93.4, 96.3]
	Replay-Mobile	10.2	5.5	7.9	92.0 [90.7, 93.2]	0.6	50.1	25.3	14.8 [13.3, 16.5]
	Combined	9.9	4.7	7.3	94.4 [93.6, 95.2]	0.5	45.0	22.8	36.1 [34.4, 37.7]

we note that both, the ISV modeling method as well as the ROC-SDK method, show levels of vulnerability significantly lower than those for the three CNN-based systems. For each dataset, the highest IAMPR values are shown in bold font in the table.

In the experiments where the vulnerability of a FR system is relatively low, the corresponding FNMR is also unacceptably high. For example, the ISV modeling method shows an IAPMR of 14.8% for the Replay-Mobile dataset, at score-threshold $\mathcal{T}_{0.1}$. The corresponding FNMR is 50.1%. A system with such a high FNMR would not be useful in practical applications.

This can also be observed from the score distributions of the FR systems. Figure 3.5 shows the score distributions of the FR systems for the combined PAD dataset. For every FR system, the score distribution of the PAs (gray histogram) significantly overlaps the genuine score distribution (green histogram). This indicates that none of the FR systems is able to adequately distinguish between genuine presentations and attack-presentations. We note, however, that both, the overlap between the PA-score-histogram and the genuine-score-histogram as well as the separation between genuine and ZEI score distributions is stronger for all the CNN-based FR systems, compared to the ISV modeling and ROC-SDK methods. The IAPMR values at $\mathcal{T}_{EER}$ shown in the plot (repeated in table 3.4) are higher for the CNN-based methods than the other two FR methods.

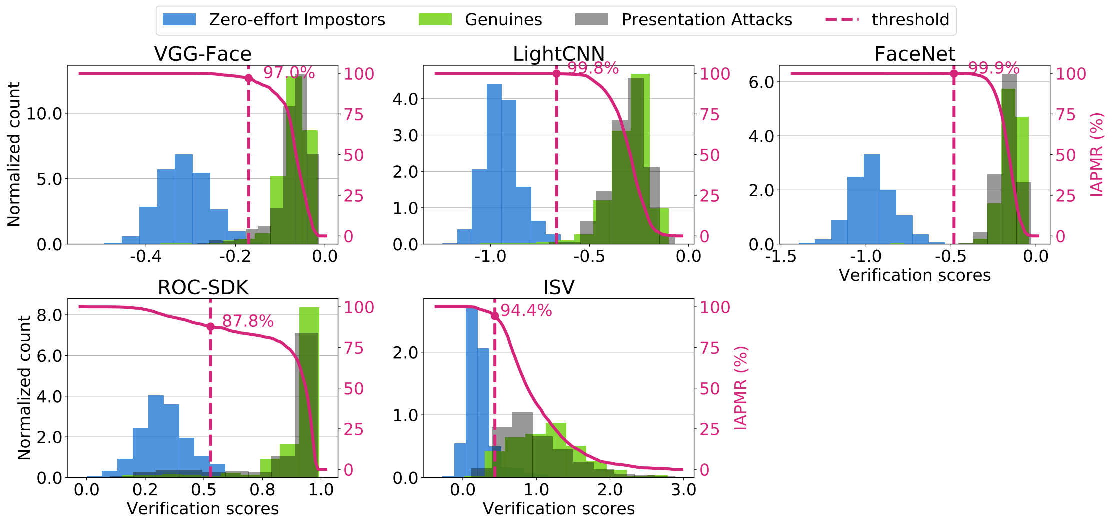


Figure 3.5 – FR score-distributions for all PAD datasets combined. Each plot corresponds to one FR method (shown above the plot), and shows three histograms of scores – for genuine presentations (green), ZEI presentations (blue) and attack presentations (gray). The red vertical dashed line marks the score-threshold $\mathcal{T}_{EER}$, corresponding the EER of the development group (of the licit protocol), for the FR method in question. Samples with scores lower than $\mathcal{T}_{EER}$ are classified as ZEI presentations. In the ideal scenario the blue histogram would lie entirely to the left of this threshold, and the green histogram would lie entirely to its right. The solid red line shows the IAPMR for different thresholds. Given a specific score-threshold, the IAPMR of resulting the FR system can be read-off at the point where the IAPMR curve (solid red curve) intersects the threshold (dashed vertical line).

The ROC-SDK method failed to generate templates for a certain number of input images, probably be due to some mechanism for rejecting low-quality input samples. This may explain the relatively lower vulnerability of this method as some PA (as well as genuine) samples may have been rejected based on their quality. We are not able to verify this hypothesis, as implementation details of ROC-SDK are not public knowledge.

Table 3.5 shows the IAPMR values for each PA type (see table 3.1 for an explanation of the various PA types in each dataset). The values in the table show the average success-rate for each type of PA, over the five FR methods. Clearly, all PA types are highly successful in spoofing all the five FR methods. The table indicates that the FaceNet CNN was successfully spoofed more often than the other FR methods.

3.3.5 Discussion of Vulnerability Analysis Results

In table 3.5 we note that PA-III attacks in the Replay-Mobile dataset are generally more successful in spoofing the FR systems than PA-III presentations in other datasets. (As described in table 3.1, PA-III refers to high-quality video replay attacks.) This observation correlates with

Table 3.5 – IAPMR (percent) of each of the three PA types listed in table 3.1.

FR Method	Dataset	PA I	PA II	PA III
VGG-Face	Replay-Attack	98.8	97.5	98.8
	MSU-MFSD	93.3	90.7	93.3
	Replay-Mobile	91		99.7
LightCNN	Replay-Attack	96.2	95.0	94.4
	MSU-MFSD	93.3	93.3	93.7
	Replay-Mobile	99.8		100
FaceNet	Replay-Attack	100	98.8	100
	MSU-MFSD	100	97.0	100
	Replay-Mobile	99.8		100
ROC-SDK	Replay-Attack	93.8	85	91.9
	MSU-MFSD	81.3	86.7	82.0
	Replay-Mobile	89.6		93.4
ISV	Replay-Attack	95	91.2	91.9
	MSU-MFSD	95.7	78.3	98.3
	Replay-Mobile	87.2		96.8

the fact that the PAI used (for video-replay attack presentations) in the Replay-Mobile dataset is of higher quality than the PAIs used for video-replay attacks in the other two, older, datasets. That is, the digital screens used as PAIs for PA-III videos in the Replay-Mobile dataset are more recent than those used for creating the older datasets.

Let us look at some specific cases of successful and unsuccessful PAs in detail. Figure 3.6 shows the most successful and the least successful PAs in attacking the VGG-Face CNN. For each PAD dataset, sample images are shown for four different clients:

- on the left, the two sets corresponding to the two clients with least successful attacks, and
- on the right, two sets of images corresponding to the two most successful PAs.

The labels on the vertical axis indicate the dataset from which the examples have been taken. For some images in the figure, the associated scores from two FR systems – the VGG-Face CNN, and the ISV modeling method – are also shown. These scores have been calibrated to a standard scale using Platt scaling (Platt et al., 1999).Therefore, the score-values shown in figure 3.6 can be seen as face-verification probabilities.

For each dataset, the label on the vertical axis of figure 3.6 also shows the calibrated score-thresholds (*i.e.,*, probability-thresholds) for the two FR systems considered in this figure. The first number gives the EER probability-threshold for the VGG-Face CNN, and the second number gives the EER probability threshold for the ISV modeling method. (The probability-thresholds are computed over the development group.) For example, the probability-threshold for the VGG-Face system, for the Replay-Attack dataset is 0.83, and the probability-threshold for the ISV modeling method, for the same dataset, is 0.44.

The figure shows three images for each client – (from left to right) an enrollment sample, a PA sample, and a genuine probe sample. For the PA and probe samples, the verification probabilities are shown in the figure, for the two FR systems (the left number is the probability assigned by the VGG-Face system for the image in question, and the number on the right is the probability estimated by the ISV modeling system). When the probability is equal to or higher than the corresponding probability-threshold, the score is shown in green (*i.e.,*, the presentation was accepted as genuine). Otherwise, the score is shown in red (*i.e.,*, the sample was rejected by the corresponding FR system).

For all three datasets, the clients in the right half of the figure are those that received the highest scores from the VGG-Face system. For these cases, both presentations (PA and genuine probe) were also scored highly by the ISV modeling method. Visual inspection of the images indicates that for all these clients, the probe and PA samples are well-cropped images, with good frontal pose, captured under reasonably good illumination conditions.

In the left half of the figure, we show the two clients, per dataset, that were assigned the lowest score by the VGG-Face system. As explained earlier, the probability values in red indicate that the sample was rejected by the corresponding FR system. For example, the PA sample of the woman in the MSU-MFSD dataset was rejected by both FR systems. The genuine probe-sample of the same client, however, was rejected by the VGG-Face system, but accepted by the ISV modeling system. This result is difficult to explain. The eyes in the three images for this client look well aligned, which indicates that the eye-localization was correct in all three images. In this case, illumination variations could be to blame for the low scores by the VGG-Face CNN. In some of the other cases we note that either the enrollment image or the probe-image is not correctly aligned to the image-axes. Note that the images shown in this figure are normalized face images. Therefore, a misaligned normalized face indicates that the eye-locations have been incorrectly estimated in the original image, during the normalization process. This error may be due to insufficient illumination, shadows, or reflections from the eye-wear.

3.4 Conclusions

For trustworthy face-biometrics systems, high face-verification accuracy and robustness to presentation attacks (PA) are equally important. In recent years deep learning based face-recognition (FR) methods have captured the interest of biometrics researchers, because they have been shown to outperform preceding methods by a wide margin. In this chapter, we have empirically verified the hypothesis that the higher the face-verification accuracy of a FR system, the higher is its vulnerability to PA. This is the first study to empirically explore the vulnerability of CNN-based FR systems to a variety of PA. For comparison, we have also included two other FR methods not explicitly based on CNNs. Specifically, we have studied the robustness of five FR methods to PA, namely, three CNN-based FR methods (VGG-Face, LightCNN and FaceNet), Inter-session Variability (ISV) modeling method, which relies on hand-crafted features, and the ROC-SDK, a commercial FR product. The three CNN-based FR methods as well as the ISV modeling method are not explicitly designed to detect PAs. (No claim can be made about the ROC-SDK as implementation details for this product are not publicly available.) Therefore, one would intuitively expect these methods to be vulnerable to PAs.

The face-verification performance and vulnerability to PA of the various FR systems are reported using recently standardized ISO metrics. Our experiments, conducted on three publicly available PAD datasets, show that, while all the studied FR systems are highly vulnerable to PAs, the three CNN-based FR methods are all more vulnerable to PAs than the other two methods (based on the IAPMR values and confidence-intervals shown in table 3.4). Among the three CNN-based methods, the FaceNet based FR method, which shows the best FR performance, is also most vulnerable to PA.

The experimental results presented in this chapter are noteworthy for the following reasons.

- Face-verification performances of the various CNN-FR systems, previously evaluated only on the LFW dataset, are confirmed using a widely used face-verification dataset (MOBIO) as well as several recent publicly available PAD datasets.
- Although the FR methods studied here have always been intuitively considered to be vulnerable to PAs, these are the first experiments to quantify this vulnerability empirically, based on large-scale study using several PAD datasets. The experiments also demonstrate that raw FR performance alone does not make a FR method suitable in face-verification applications.
- Our experiments clearly demonstrate that FR methods with higher FR accuracy also show higher levels of vulnerability to PA.
- Since these experiments have involved publicly available PAD datasets, the numerical results presented here can serve as baseline performance-values for researchers in the field of face PAD.

These results make a clear case for incorporating explicit countermeasures to make FR systems significantly robust to PAs. In future work, we will study the impact of combining these FR methods with a PAD system.

The FR systems studied in this work have all been trained only for face-recognition. The next logical step is to expressly take PAs into account when training the FR systems. Such experiments would have greatly enlarged the scope of the current work. In a subsequent work we plan to show results of training a CNN for FR, when PAs are explicitly identified as a separate class.

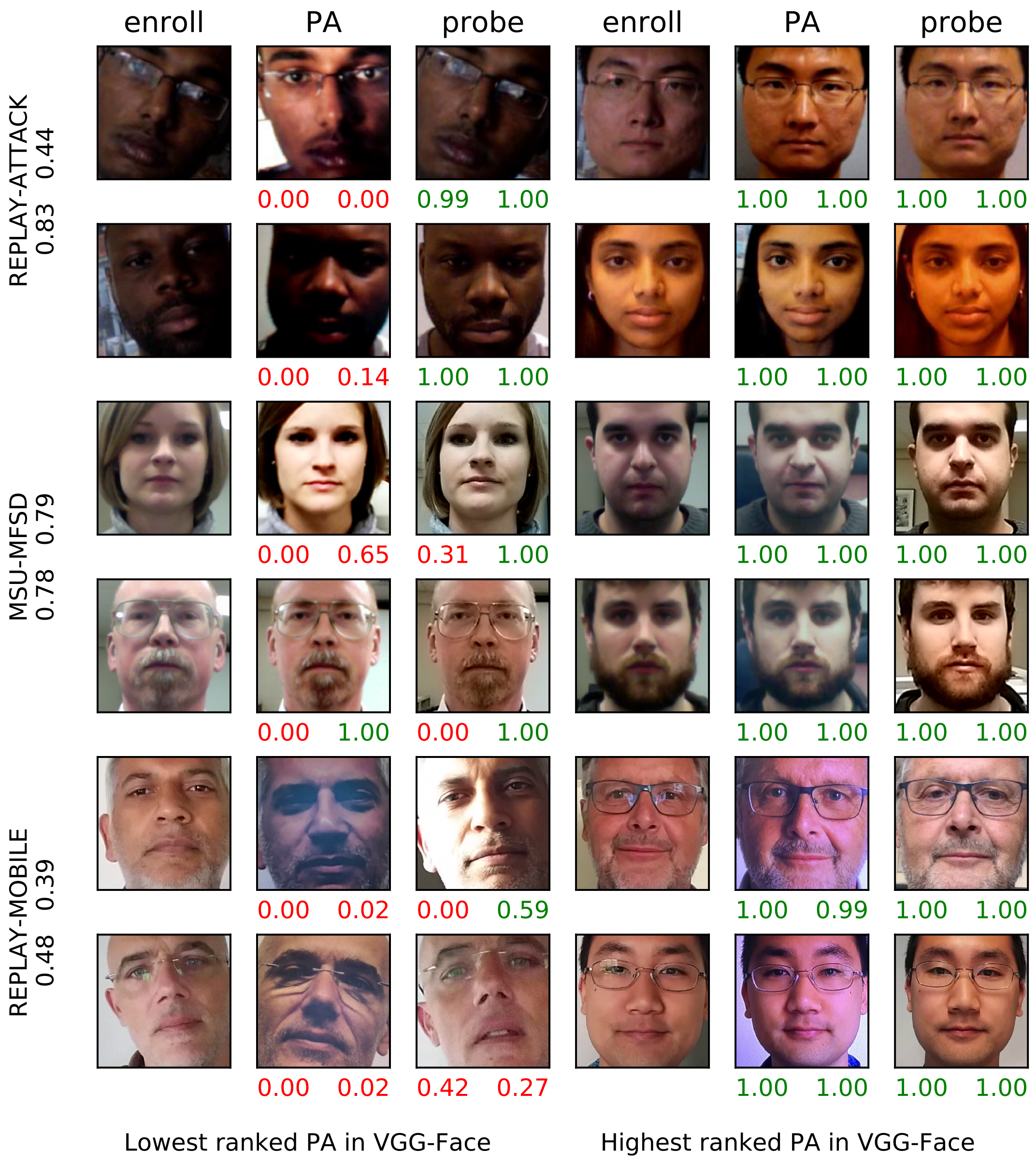


Figure 3.6 – Examples of unsuccessful and most successful PAs in attacking the VGG-Face system. The three images on the left are associated with the unsuccessful PAs and the three images on the right are associated with the most successful PAs. Two sets of examples (two rows) is shown from each dataset. Three images are shown for each client: (left) an enrollment sample, (middle) example PA, and (right) the genuine probe image that has the closest score to that of the corresponding PA sample. For each dataset, the EER probability-thresholds are shown on the y-axis label for the VGG-Face and ISV systems, respectively. For each PA sample and probe sample, verification-probabilities assigned by the VGG-Face system (left) and the ISV modeling system (right) are indicated below the sample. The verification-probability is red if a sample is rejected using the EER probability-threshold and green otherwise.

4 Evaluation of the Generalization of CNN-Based Face PAD

CNNs (see sections 2.1 and 2.2 on pages 9 and 14 for a background on CNNs) are shown to be successful in solving many computer vision problems such as image classification (Krizhevsky, Sutskever, and G. E. Hinton, 2012) and object detection (K. He et al., 2016). They have also been applied on the problem of face PAD. The main advantage of using CNNs for face PAD, compared to using hand-crafted features, is that they *learn* filters which extract features from face images that are appropriate for PAD. There has been many CNN architectures proposed for face PAD (J. Yang, Lei, and S. Z. Li, 2014; Menotti et al., 2015; Xu, S. Li, and Deng, 2015; Lei Li et al., 2016; K. Patel, Hu Han, and Jain, 2016; Z. Boulkenafet et al., 2017; Atoum et al., 2017; Xiaokang Tu and Fang, 2017; H. Li, P. He, et al., 2018; Ying, X. Li, and Chuah, 2018; Hao and Pei, 2019; Jaiswal et al., 2019; L. Li et al., 2019; George and Marcel, 2019; Almeida, 2018; Xiaoguang Tu, J. Zhao, et al., 2019). Some of these methods were detailed in section 2.4.

However, no systematic evaluation of the generalization of these CNNs in cross-dataset scenarios has been done before. In this chapter, performance of several face PAD CNNs is evaluated in terms of both intra-dataset and cross-dataset performance. This will serve as a baseline evaluation for the following chapters in this thesis. Moreover, an evaluation methodology is adopted which allows fair comparison of these architectures by partially avoiding the dataset bias present in face PAD datasets.

This chapter is organized in the following way. Details of the CNN architectures used in the experiments are given in section 4.1. The evaluation methodology and experimental results are presented in section 4.2. Finally, the results are discussed in section 4.3.

4.1 Architectures

The first work using CNNs for face PAD (J. Yang, Lei, and S. Z. Li, 2014) (the method was detailed in section 2.4.1 on page 36) used normalized face images (see section 2.3.1 on page 28 for face normalization) with a large amount of visible background as input to these architectures (see scales of 4 and 5 in figure 2.12). However, including background in the processed images may lead to bias in trained models as a background which gives hints that an image is a PA can be avoided by an attacker (see section 2.4.1 on page 36 for more explanation). Moreover, in face PAD, we are interested in some cues in images that can help us identify an image as a PA (Marcel et al., 2019). Some of these cues might be present in all parts of a face image. For example, when an image is printed and is presented to a camera, it may contain certain noise patterns that is specific to printers. Similarly, digital displays may show certain noise patterns when used as PAIs. Therefore, a face PAD CNN may use only a **patch** of the face image as input instead of the whole face image and classify each patch as a PA or *bona fide* independently of other parts of the face image.

The following CNN architectures are evaluated for face PAD in this chapter. The **MSU-Patch** (Atoum et al., 2017), **DeepPixBiS** (George and Marcel, 2019) architectures are face PAD baselines which are reproduced in this work. The **InceptionResNetV2** (Szegedy, Ioffe, et al., 2016) architecture was tested on FR in (Pereira, Anjos, and Marcel, 2019). In this work, the **InceptionResNetV2** is used with the same parameters as in (Pereira, Anjos, and Marcel, 2019) for face PAD. The **SimpleCNN** architecture is a small patch-based face PAD CNN that will be used as a baseline as well. The **InceptionResNetV2-SimpleCNN** and **MultiScale-InceptionResNetV2** architectures are novel architectures which are proposed in this thesis. All the architectures will be described in the following sections.

These architectures are different from each other in terms of several criteria:

- the input may be a face image or a patch of the face image or both,
- the resolution of input images may be different,
- the face normalization process may be different,
- the number of parameters of the networks may be different,
- their output may be different, some may output one value for the whole face image and some may output a map of decisions for each patch of the face image, and,
- they might use different losses and data augmentation schemes during training.

The differences are detailed below and are summarized in table 4.1.

These architectures work on still color RGB images and the final score of a video of face images is the average score of tested frames. For all the architectures except the MSU-Patch architecture, faces are geometrically normalized so that the eye centers fall on predetermined locations in the final image (see section 2.3.1 and figure 2.10 on page 28). For the MSU-Patch architecture, as it was done by its authors (Atoum et al., 2017), face images are cropped out of the original size images without any transformation. Then, random 96×96 patches are extracted from the face image and are used as input to the system. In this work, only two random patches are extracted to keep the computations tractable because the original

Table 4.1 – Differences between the CNN-based face PAD architectures used in the experiments. See section 4.1 for details.

	MSU-Patch (baseline)	**SimpleCNN (baseline)**	**InceptionResNetV2 (baseline)**	**InceptionResNetV2 and SimpleCNN**	**DeepPixBiS (baseline)**	**MultiScale-InceptionResNetV2**
Input format	Patch	Patch	Face	Face and Patch	Face	Face
Input resolution	96x96	28x28	160x160	160x160 and 28x28	224x224	224x224
Number of parameters	3.5 million	3.2 million	57 million	60.2 million	3.2 million	2.6 million
Face image normalization	Just cropping the face region	Geometric face normalization	Geometric face normalization	Geometric face normalization	Geometric face normalization	Geometric face normalization
Output shape	2	2	2	2	14x14x1	13x13x1
Output activation	softmax	softmax	softmax	softmax	sigmoid	sigmoid
Data augmentation	random patches	random patches	random crop	random crop	random crop random brightness random contrast random saturation random horizontal flip	random gamma random crop random brightness random contrast random saturation random horizontal flip scale jitter
Loss	cross-entropy	cross-entropy	cross-entropy	cross-entropy	pixel-wise cross-entropy	pixel-wise cross-entropy

resolution of images of the datasets used in this work is quite high. While Atoum et al. (2017) argue that avoiding any transformation on the face images keeps the noise patterns of PAs untouched, this makes the MSU-Patch architecture sensitive to the original resolution of input images. For the SimpleCNN architecture, non-overlapping 28×28 color patches are extracted from normalized 160×160 face images. For the InceptionResNetV2 architecture, 160×160 face images are used as input as it was done in (Pereira, Anjos, and Marcel, 2019) when the architecture was used for FR. For the InceptionResNetV2-SimpleCNN architecture, which is made of both InceptionResNetV2 and SimpleCNN architectures, the input to the individual architectures are the same as InceptionResNetV2 and SimpleCNN architectures. The input for the DeepPixBiS and MultiScale-InceptionResNetV2 architectures is 224×224 pixel normalized face images.

All architectures are trained using **cross-entropy** as the loss-function:

$$H_{\boldsymbol{y}'}(\boldsymbol{y}) = -\sum_{(x_i,\boldsymbol{y}'_i)}^{\mathbb{X}} \sum_{k=1}^{K} y'_{ik} log(y_{ik}) \tag{4.1}$$

where K is the total number of classes (2 for PAD), x_i is the ith sample from a set of samples $\mathbb{X}$, $\boldsymbol{y}'_i$ is the **one-hot encoded** true label of x_i where y'_{ik} is 1 and other values of $\boldsymbol{y}_i$ is 0 if sample x_i is from kth class, and y_{ik} is the predicted probability of the model of the sample x_i belonging to class k. For the DeepPixBiS and MultiScale-InceptionResNetV2 architectures which output a map of probabilities, **pixel-wise cross-entropy** is used as the loss-function which is the average cross-entropy values over each pixel of the output map. If the final layer of the architectures has one output value, the sigmoid function (see section 2.1 on page 9) is used to normalize the output between 0 and 1 (so that they can be treated as probabilities). When the final layer of the architecture outputs two values, the softmax function is used to normalize the output values between 0 and 1.

During training, several **data augmentation** methods can be used to increase the variability of the training data which in turn may improve the generalization performance of the models. The data augmentation methods that were used are described below and are visualized in figure 4.1.

- **random crop**: where a part of the face-image is randomly cropped and is used as input instead of the whole image. When random crop is applied, the normalized face images have a slightly bigger size than the expected input size so that the randomly cropped image has the expected input size.
- **random patches**: where a patch of the face-image is cropped from a *random location* in the image, *i.e.,* the patches are not extracted from pre-determined locations.
- **random brightness**: where the brightness of the image is changed randomly.
- **random contrast**: where the contrast of the image is changed randomly.
- **random saturation**: where the saturation of the image is changed randomly.
- **random gamma**: where Gamma correction[1] with random Gamma values is applied on the image.
- **random horizontal flip**: where some images are flipped alongside their width randomly.
- **scale jitter**: where the images are randomly scaled up or down and then randomly cropped or padded with zeros so that their resolution remains constant.

The CNN architectures that were used in the experiments are detailed below.

4.1.1 MSU-Patch (baseline)

This architecture is made of 5 convolutional layers and 2 fully-connected layers. The input is 96 × 96 pixel patches extracted from original size face images. The output is the probability of the patch being a PA. At test time, two random patches is used for scoring. The final score for the face image is the average probability of all tested patches. The network is trained using the cross-entropy loss-function. For more details on this baseline, see section 2.4.2 on page 37.

4.1.2 DeepPixBiS (baseline)

This architecture is composed of the first eight blocks of DenseNet 161 (G. Huang et al., 2017) (all initial layers till *transition_block_2*, see section 2.2.8 and table 2.8 on pages 21 and 24) and another convolutional layer which acts as a classifier. The architecture outputs a 14 × 14 pixels map given a 224 × 224 face image as input. Each *pixel* in the output map contains a probability of the face image being a PA and may report a different probability because each pixel focuses

[1] https://en.wikipedia.org/wiki/Gamma_correction

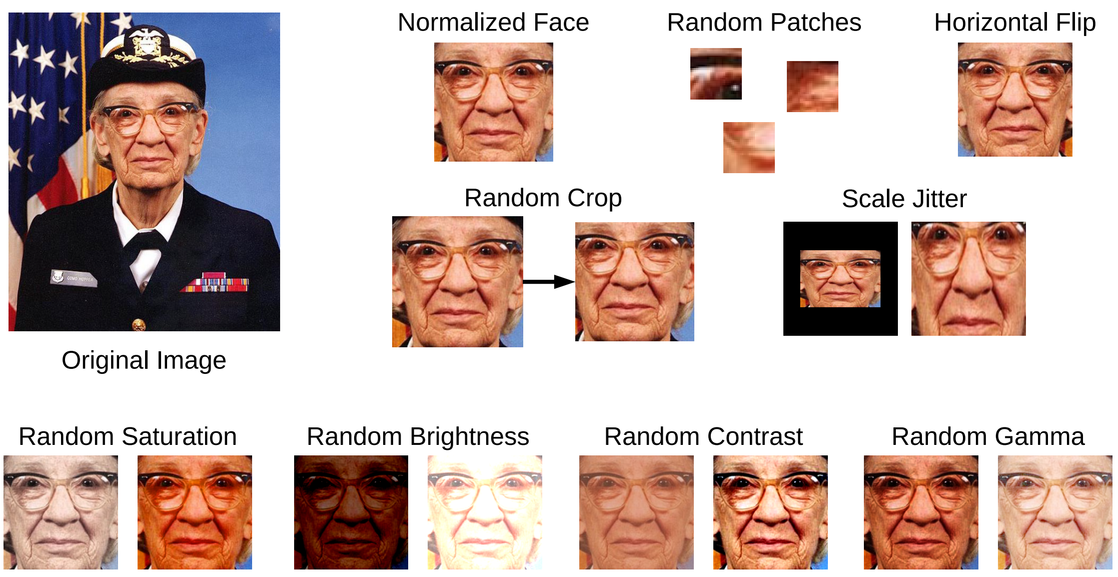


Figure 4.1 – Visualization of different data augmentation methods used during the training of the CNN models. Note that the degree of some of the data augmentation methods have been increased in the figure to make the changes more apparent in the figure. In the experiments, the changes in brightness, contrast, *etc.* are not as drastic as shown here.

on a different part of the face image. The output map is averaged to report one probability given a face image. The network is trained using the *pixel-wise cross-entropy* loss-function. For more details on this baseline, see section 2.4.3 on page 38.

4.1.3 InceptionResNetV2 (baseline)

The InceptionResNetV2 architecture (Szegedy, Ioffe, et al., 2016), detailed in section 2.2.7, has shown promising performance on a range of visual recognition tasks including face recognition[2] (Pereira, Anjos, and Marcel, 2019). Here, this architecture is used for face PAD using the same setup as in (Pereira, Anjos, and Marcel, 2019). The input to the CNN is a 160×160 pixels face image and the output is the probability of the face image being a PA. The network is trained using the cross-entropy loss-function.

4.1.4 SimpleCNN (baseline)

The **SimpleCNN** architecture is a very small CNN that uses recent common techniques in building CNNs such as using batch normalization and dropout. The architecture is made of two convolutional layers and two fully-connected layers. The architecture is detailed in table 4.2. The input is 28×28 pixel patches extracted from 160×160 normalized face images. The output is the probability of the patch being a PA. At test time, all non-overlapping patches from the face image are extracted and the probability of each patch being a PA is reported. The

[2]https://www.idiap.ch/software/bob/docs/bob/bob.bio.face_ongoing/v1.0.4/leaderboard.html

Table 4.2 – Details of the SimpleCNN architecture. The input to the network is 28 × 28 patches of face images. Face images are normalized to 160 × 160 pixels.

Layer (type)	Details	Output Shape	Number of Parameters
conv2d (Conv2D)	filters=32, kernel_size=(3, 3), strides=(1, 1), padding=same	(28, 28, 32)	864
batch_normalization (BatchNormalization)		(28, 28, 32)	96
activation (Activation)	activation=relu	(28, 28, 32)	0
max_pooling2d (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=same	(14, 14, 32)	0
conv2d_1 (Conv2D)	filters=64, kernel_size=(3, 3), strides=(1, 1), padding=same	(14, 14, 64)	18,432
batch_normalization_1 (BatchNormalization)		(14, 14, 64)	192
activation_1 (Activation)	activation=relu	(14, 14, 64)	0
max_pooling2d_1 (MaxPooling2D)	pool_size=(2, 2), strides=(2, 2), padding=same	(7, 7, 64)	0
flatten (Flatten)		(3136)	0
dense (Dense)	units=1024	(1024)	3,211,264
batch_normalization_2 (BatchNormalization)		(1024)	3,072
activation_2 (Activation)	activation=relu	(1024)	0
dropout (Dropout)	drop_rate=0.4	(1024)	0
dense_1 (Dense)	units=2, activation=softmax	(2)	2,050
Model	Parameters: total=3,235,970, trainable=3,233,730		

final probability of the face image being a PA is the average of the probabilities of the patches. This architecture was proposed to see if very small CNNs (in terms of depth) are able to detect PAs or deeper architectures are needed.

4.1.5 InceptionResNetV2-SimpleCNN (proposed)

While patch-based CNNs have shown to be effective for detecting PAs (Atoum et al., 2017), it is reasonable to assume that some holistic cues, such as the face shape, are important as well in detecting PAs. For example, when faces are printed on paper, the paper may bend which causes the face to be skewed in the captured image. Therefore, using cues from *both* patches and the whole face may improve the performance of face PAD systems. The **InceptionResNetV2-SimpleCNN** architecture is proposed to take advantage of both patch-based and whole face-based cues.

The architecture is made of the two InceptionResNetV2 and SimpleCNN architectures to create a larger architecture that combines a patch-based CNN and a whole face-based CNN. This allows the final CNN to output one probability given a face image while considering both the whole face and patches. The diagram of the proposed architecture is shown in figure 4.2 and is detailed below.

Given a 160 × 160 pixels face image as input, $\mathbf{I}$, the output of the final architecture, $\boldsymbol{y}$ is calculated in the following way:

$$\boldsymbol{y} = \text{Dense}_{\text{units=2, activation=softmax}}([\boldsymbol{e}_1, \boldsymbol{e}_2]) \tag{4.2}$$

$$\boldsymbol{e}_1 = \text{InceptionResNetV2}_{\text{avg\_pool}}(\mathbf{I}) \tag{4.3}$$

$$\boldsymbol{e}_2 = \frac{1}{P}\sum_{i=1}^{P} \text{SimpleCNN}_{\text{dropout}}(\mathbf{I}_i) \tag{4.4}$$

where [...] is the concatenation of vectors to form a larger vector, Dense is a fully-connected layer with two output units and softmax activation (see section 2.1 on page 9), $\boldsymbol{e}_1$ is the face embedding of InceptionResNetV2 at the *avg_pool* layer (one layer before the output, see table 2.7 on page 23), $\mathbf{I}_i$ is the ith 28×28 patch of the image from the total P non-overlapping[3] patches extracted from $\mathbf{I}$, and $\boldsymbol{e}_2$ is the average patch embedding of SimpleCNN at the *dropout* layer (one layer before the output, see table 4.2). Note that the total number of non-overlapping patches, P, here 25, depends on the resolution of the input image, $\mathbf{I}$, here 160×160, and the resolution of patches, here 28×28.

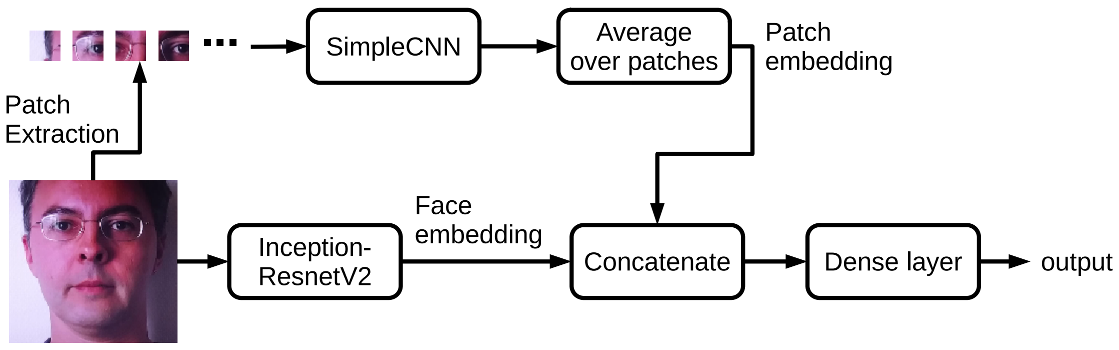


Figure 4.2 – Diagram of the proposed InceptionResNetV2-SimpleCNN CNN architecture. The input face image is given to the InceptionResNetV2 architecture and its patches are given to the SimpleCNN architecture. The embeddings of SimpleCNN is averaged over all patches to make a final embedding for all patches. The face and patch embeddings are concatenated to make a larger embedding and is used as input to a fully-connected layer which outputs one probability given a face image.

4.1.6 MultiScale-InceptionResNetV2 (proposed)

The proposed **MultiScale-InceptionResNetV2** CNN architecture classifies a face image using feature maps from different scales. In a CNN architecture, the input image is processed into feature maps and some layers downsample the feature maps (such as pooling layers). Hence, a CNN architecture produces feature maps of different scales at different layers. Feature maps at different scales might all prove to be useful for PAD because most of the time in face PAD we are interested in the *noise patterns* (*e.g.,*the moiré pattern present in replay PAs using a digital display as PAI) that is present in face images as cues for detecting PAs. These noise patterns may be present in different scales in the final normalized face image depending on the distance of the camera to the PAI. To explicitly use feature maps of different scales in the final decision of the face PAD CNN, the MultiScale-InceptionResNetV2 CNN architecture concatenates feature maps of *three* different scales and uses those as embeddings for face PAD. This architecture is detailed in figure 4.3 and table 4.3.

The architecture uses the feature maps at the output of the *block35_3* layer as the first scale (after being down-sampled twice with pooling layers to match the resolution of the third scale), the output of the *block17_3* layer as the second scale (after being down-sampled once with a

[3]For simplicity, non-overlapping patches are used. Otherwise, overlapping patches may be used as well where each patch overlaps by a few pixels with its neighbor patches.

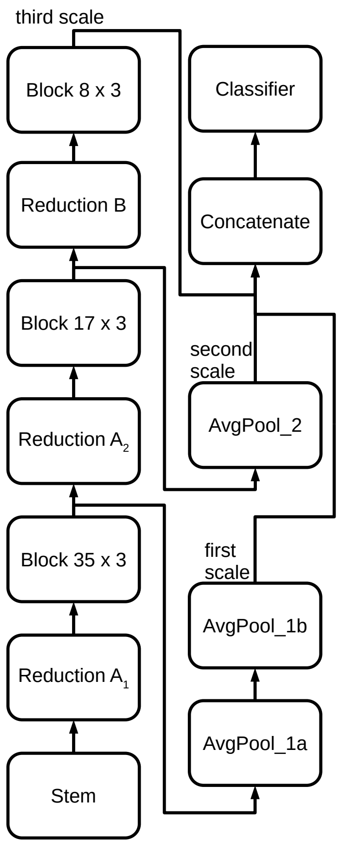


Figure 4.3 – Diagram of the MultiScale-InceptionResNetV2 architecture. The components are detailed in table 4.3. The classifier block consists of all the layers that are placed after the *concatenate* layer (see table 4.3).

pooling layer to match the resolution of the third scale), and the output of the *block8_3* layer as the third scale. The feature maps are concatenated channel-wise (at layer *concatenate*) to form a feature map with features from three different scales. This feature map is used as input to a few *classification* layers to make a final decision. Similar to DeepPixBiS, a pixel-wise output map probability is outputted given an input face image of 224×224 pixels. The pixel-wise cross-entropy loss-function is used for training the network. Moreover, *scale jitter* data augmentation is used in the training of the network to improve the generalization of the network on different scales.

Below, the evaluation methodology, experiments, and discussions are presented in the following sections.

4.2 Experiments and Analysis

When evaluating data-driven PAD systems, the database design is important to make sure the evaluations are not biased. Therefore, an evaluation methodology, presented in section 4.2.1,

Table 4.3 – Details of the MultiScale-InceptionResNetV2 architecture. The diagram of this architecture is shown in figure 4.3. The components used in this architecture are the same components of the InceptionResNetV2 architecture which are detailed in section 2.2.7.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
Stem (Conv2D_BN)	filters=4, kernel_size=1, strides=1	(224, 224, 4)	24	input
Reduction_a_1 (ReductionA)	k=8, kl=12, km=14, n=14	(111, 111, 32)	3,056	Stem
block35_1 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=2	(111, 111, 32)	16,648	Reduction_a_1
block35_2 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=2	(111, 111, 32)	16,648	block35_1
block35_3 (InceptionResnetBlock)	block_type=block35, scale=0.17, n=2	(111, 111, 32)	16,648	block35_2
Reduction_a_2 (ReductionA)	k=32, kl=32, km=32, n=32	(55, 55, 96)	29,056	block35_3
block17_1 (InceptionResnetBlock)	block_type=block17, scale=0.17, n=2	(55, 55, 96)	124,496	Reduction_a_2
block17_2 (InceptionResnetBlock)	block_type=block17, scale=0.17, n=2	(55, 55, 96)	124,496	block17_1
block17_3 (InceptionResnetBlock)	block_type=block17, scale=0.17, n=2	(55, 55, 96)	124,496	block17_2
Reduction_b (ReductionB)	k=64, kl=72, km=80, n=64, no=96, p=64, pq=72	(27, 27, 344)	210,048	block17_3
block8_1 (InceptionResnetBlock)	block_type=block8, scale=0.17, n=1	(27, 27, 344)	590,200	Reduction_b
block8_2 (InceptionResnetBlock)	block_type=block8, scale=0.17, n=1	(27, 27, 344)	590,200	block8_1
block8_3 (InceptionResnetBlock)	block_type=block8, scale=0.17, n=1	(27, 27, 344)	590,200	block8_2
AvgPool_1a (AveragePooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(55, 55, 32)	0	block35_3
AvgPool_1b (AveragePooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(27, 27, 32)	0	AvgPool_1a
AvgPool_2 (AveragePooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(27, 27, 96)	0	block17_3
Concatenate (Concatenate)	axis=3	(27, 27, 472)	0	AvgPool_1b AvgPool_2 block8_3
Conv2d_1 (Conv2D_BN)	filters=256, kernel_size=1, strides=1	(27, 27, 256)	121,600	concat
AvgPool_3 (AveragePooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(13, 13, 256)	0	Conv2d_1
Conv2d_2 (Conv2D_BN)	filters=128, kernel_size=1, strides=1	(13, 13, 128)	33,152	AvgPool_3
Dropout (Dropout)	drop_rate=0.2	(13, 13, 128)	0	Conv2d_2
Pixel_Logits (Conv2D)	filters=1, kernel_size=(1, 1), strides=(1, 1), padding=same	(13, 13, 1)	129	Dropout
Model	Parameters: total=2,591,097, trainable=2,581,025			

is adopted to avoid biases as much as possible. Then, the experiments are presented in section 4.2 and results are discussed in section 4.3.

4.2.1 Evaluation Methodology

Data-driven PAD systems usually require two distinct sets of data for training and hyper parameter selection: a *training set* and a *development set*. The training set is used to learn the parameters of the model and the development set is used to select the appropriate values for hyper parameters. In neural network based PAD systems, where the parameters are learned using a loss-function and back propagation, a stopping criteria is needed to stop the training. The accuracy of the PAD system on the development set may be used as a metric to stop the training process. The model that has the highest accuracy on the development set is chosen for further evaluations. This method, known as **early-stopping**, avoids overfitting in neural networks (Goodfellow, Y. Bengio, and Courville, 2016).

However, the design of PAD datasets plays an important role here. When designing the protocols of a dataset, we want to have the training set and the development set of the dataset to differ in several covariate factors which might change between the training set and the evaluation set (similar to real-world evaluations). In other words, if we assume that factors such as identity, lighting condition, biometric sensor, and PAI are covariate factors in face PAD systems, then, we want to vary these factors between the training and development sets. This way, the generalization of a model to *unseen* variations of covariate factors can be assessed.

Unfortunately, this is not true in the current face PAD datasets. In most face PAD datasets, usually mainly identities change between the training and development sets and other covariate factors are kept constant. For example, in protocol 1 of OULU-NPU (see section 2.7.5 on page 60 for details on OULU-NPU), which is one of the datasets that is often used for evaluating face PAD baselines (Z. Boulkenafet et al., 2017), the only covariate factor that changes between the training and development set is identities and all other covariate factors such as the lighting conditions are kept constant between the two sets. For the evaluation set, however, the lighting condition is different from the training and development sets. The fact that only the identities change between the training and development sets, can easily lead to overfitting of neural networks as we shall see in the experiments of section 4.2.

Because *all* the face PAD baselines of this thesis are trained on Protocol 1 of OULU-NPU (which is one of the most challenging face PAD datasets available today), this can easily bias our conclusions. We might observe a system perform poorly on the evaluation set of OULU-NPU (or another dataset), compared to other systems, only because the early-stopping method is not implemented correctly. To avoid this issue, in this thesis, the development set of another dataset (arbitrarily, the SWAN dataset) is used in the early-stopping implementation of the methods. This is especially important in cross-dataset evaluations where most covariate factors change between datasets.

To summarize, in all experiments (unless specified otherwise), all models are trained on the OULU-NPU dataset (Protocol 1 and only the training set is used) and the best model during training is chosen based on their accuracy on the development set of the SWAN dataset (Protocol 'pad_p2_face_f1', see section 2.7.6 on page 61). This makes sure almost all covariate factors are different between the training set and the development set.

4.2.2 Performance Evaluation Metrics and Plots

In this chapter and the following chapters, the methods are compared using receiver operating characteristic (ROC) plots and area under the curve (AUC) of the ROC plots. The overview of the ROC plots and AUC metrics are given below (see section 2.6.1.2 on page 52 for background on ROC and AUC).

The ROC curves are computed with $APCER$ (*log-scale*) along the x-axis and $1 - BPCER$ on the y-axis. The AUC values are computed from the same log-scale ROC plots. Because the ROC plots are plotted in *log-scale*, the AUC values can be higher than 1.

All ROC plots come in pairs where they show the performance of the method on the development and evaluation sets of the test dataset. Based on the scores of samples in the development set, an operating threshold is chosen by fixing an APCER value (a vertical line in the ROC plot). Then, this operating threshold is used to compute the errors (APCER and BPCER) of the system when tested on the evaluation set. These errors correspond to a point in the ROC plot of the evaluation set.

Moreover, because all the evaluated face PAD systems in this thesis are trained on OULU-NPU only, intra-dataset evaluations happen when the systems are tested on OULU-NPU. When the systems are tested on another dataset (Replay-Mobile, SWAN, and WMCA), the evaluations represent *cross-dataset* evaluations.

4.2.3 Experiments

There are two sets of experiments presented in this section. First, the effectiveness of the adopted evaluation methodology (see section 4.2.1) is compared to a traditional evaluation methodology. Then, the performance of the face PAD CNN architectures are compared in terms of both intra-dataset and cross-dataset evaluations.

4.2.3.1 The effect of adopted evaluation methodology

All the discussed face PAD CNN architectures are evaluated twice in this section. First, using the *traditional* methodology where the models are trained on one dataset (OULU-NPU) and the training is stopped based on the accuracy on the development set of the same dataset (OULU-NPU). Second, using the *adopted* methodology in section 4.2.1 where the CNNs are still trained on OULU-NPU but the training is stopped based on the accuracy on the development set of the SWAN dataset.

The results of evaluations are shown in figure 4.4 in terms of AUC. In the plots, each system is trained using both the traditional and adopted evaluation methodologies and they are marked with numbers 1 and 2, respectively. In one plot of figure 4.4, each two system with the same color may be compared with each other. Evaluations on OULU-NPU are considered intra-dataset evaluations and evaluations on other datasets are considered cross-dataset evaluations.

The following observations can be drawn from figure 4.4. In terms of intra-dataset performance: no significant changes in performance were observed in *DeepPixBiS* and *SimpleCNN*, performance of *MultiScale-InceptionResNetV2*, *InceptionResNetV2*, and *MSU-Patch* improved significantly, and performance of *InceptionResNetV2-SimpleCNN* degraded significantly when trained using the adopted method compared to the traditional method. In terms of cross-dataset performance: performance of *DeepPixBiS*, *InceptionResNetV2*, and *MSU-Patch* increased significantly, performance of *SimpleCNN* did not change significantly, and performance of *MultiScale-InceptionResNetV2* and *InceptionResNetV2-SimpleCNN* degraded significantly when trained using the adopted method compared to the traditional method.

Given that, using the adopted evaluation methodology, compared to the traditional methodology, the performance of five systems out of six systems either did not change or improved in intra-dataset evaluations and performance of four systems out of six systems did not change or improved in cross-dataset evaluations, the adopted evaluation methodology is used for the rest of the experiments in this chapter and the following chapter. Also, we can hypothesize that some of these networks are suffering from over-fitting due to the poor design of face PAD datasets.

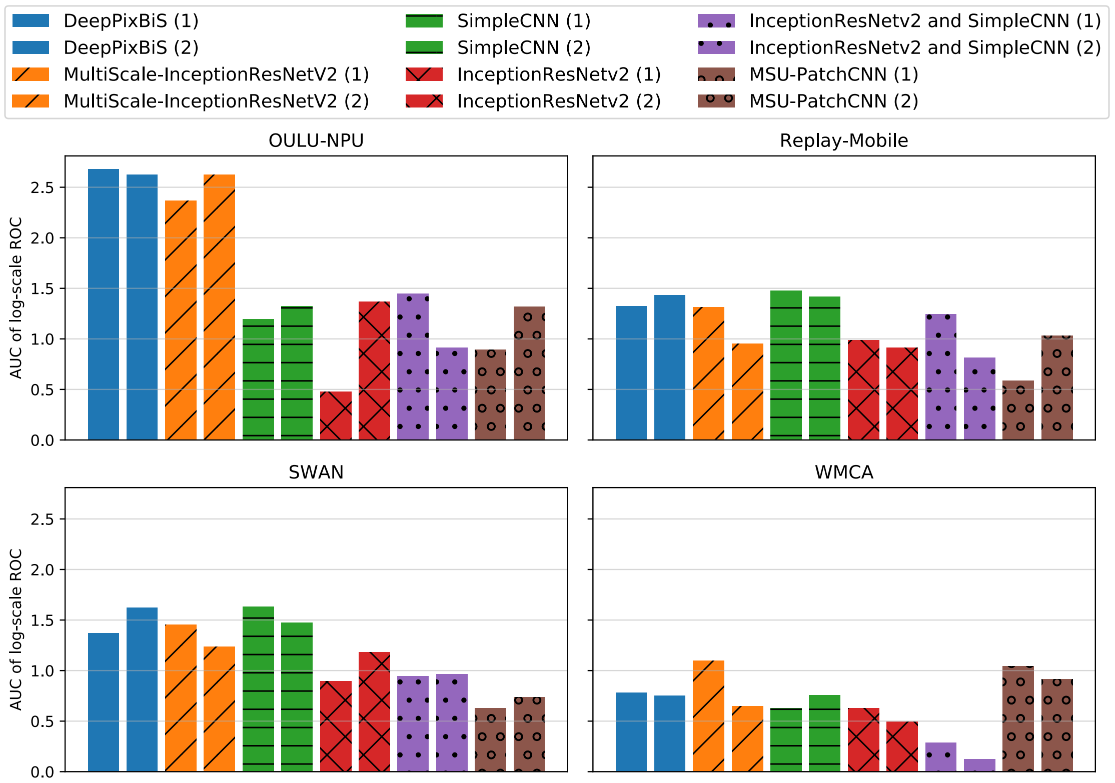


Figure 4.4 – Comparison of evaluations methodologies discussed in section 4.2.1 in terms of AUC of *log-scale* ROCs (see section 4.2.2). All models are trained using the training set of OULU-NPU. The models that their training is stopped based on the accuracy on the development set of OULU-NPU are marked with (1) and when accuracy on the development set of SWAN is used to stop the training of the models, models are marked with (2). The models are evaluated on four datasets: OULU-NPU, Replay-Mobile, SWAN, and WMCA. The dataset that is used for evaluation is mentioned in the title of the plots.

4.2.3.2 Performance Comparison of CNN-based face PAD

To compare the performance of CNN architectures in face PAD, ROC plots of evaluations (using the adopted evaluation methodology of section 4.2.1) are shown in figure 4.5. Based on the performance of the CNN architectures on the evaluation set of each dataset, we may observe the following:

- in intra-dataset evaluations (when the models are trained and evaluated on the OULU-NPU dataset), the *DeepPixBiS* and *MultiScale-InceptionResNetV2* systems *significantly* outperform other systems.

- in Replay-Mobile evaluations (cross-dataset evaluation), the *DeepPixBiS* and *SimpleCNN* systems significantly outperform other systems,

- in SWAN evaluations (cross-dataset evaluation), the *DeepPixBiS* system outperforms other systems. Although, in APCER values smaller than 0.01 (1%), the *SimpleCNN* and *InceptionResNetV2* systems outperform the *DeepPixBiS* system.

- in WMCA evaluations (cross-dataset evaluation), all systems have a poor performance compared to other cross-dataset evaluations and have similar performance. The *MSU-Patch*, *DeepPixBiS*, *MultiScale-InceptionResNetV2*, and *SimpleCNN* perform slightly better compared to other systems.

Overall, *SimpleCNN*, *DeepPixBiS* and *MultiScale-InceptionResNetV2* architectures (which either use a patch as input or use a pixel-wise cross-entropy loss-function) are promising face PAD CNN architectures according to our intra-dataset and cross-dataset evaluations.

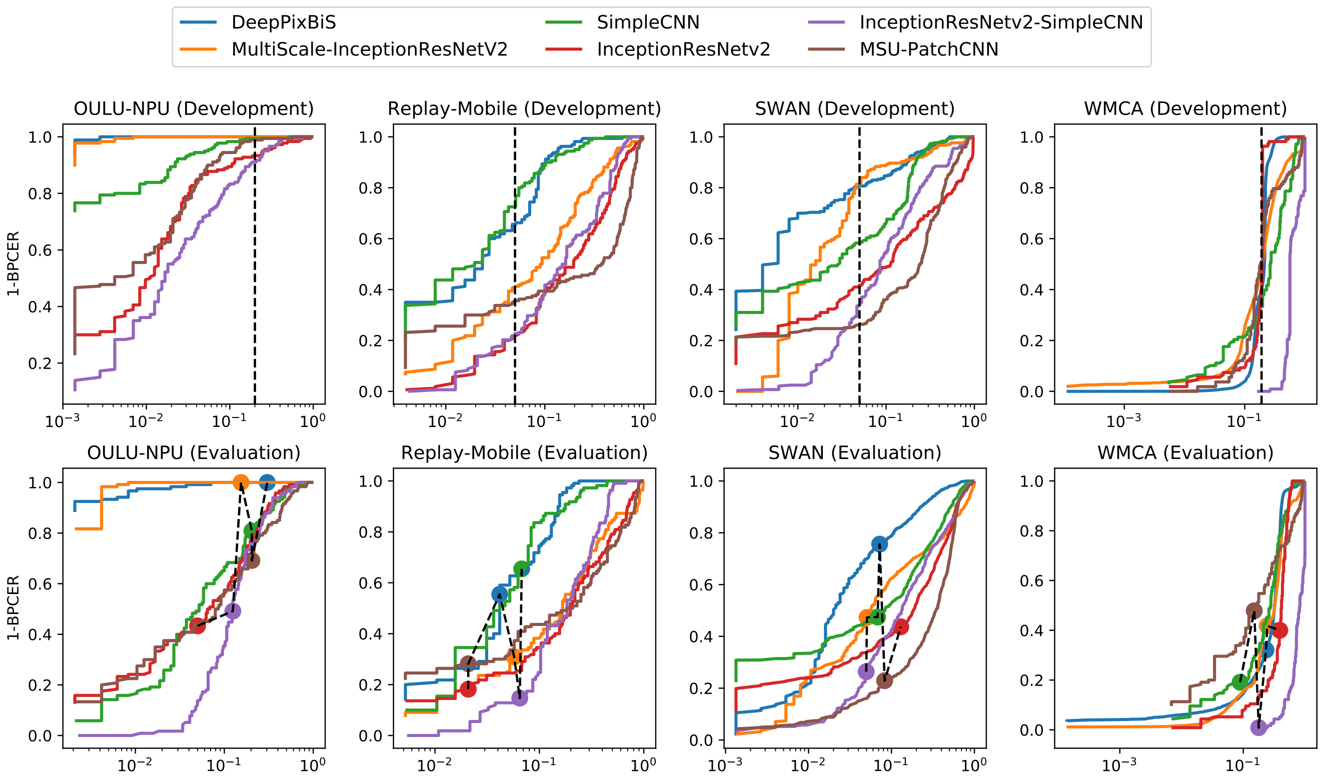


Figure 4.5 – Evaluation, in terms of ROC plots, of multiple face PAD CNN architectures on multiple datasets is shown. All architectures are trained using the adopted evaluation methodology in section 4.2.1. To learn how to read ROC plots, see section 4.2.2.

4.3 Conclusions

In this chapter, four baseline CNNs (SimpleCNN, MSU-Patch, DeepPixBiS, and InceptionRes-NetV2) and two proposed CNNs (InceptionResNetV2-SimpleCNN, and MultiScale-InceptionResNetV2) were evaluated on the problem of face PAD in both intra-dataset and cross-dataset scenarios. The SimpleCNN, MultiScale-InceptionResNetV2, and DeepPixBiS architectures showed promising performance in both intra-dataset and cross-dataset evaluations. Training face PAD CNNs on patches of face-images (instead of whole face images) or outputting pixel-wise probability maps (instead of just one probability for the whole face image), are promising approaches in improving the performance of face PAD CNNs. Although, more investigation in this direction is needed to clearly understand why some CNN architectures perform better compared to other architectures in face PAD. All in all, the DeepPixBiS architecture proposed by George and Marcel (2019) showed consistent significant improvements in terms of both intra-dataset and cross-dataset evaluation settings compared to other methods.

Moreover, it was shown that the variability of samples in the training and development sets of protocol 1 of the OULU-NPU dataset is limited which may lead to overfitting in training of the neural networks. This overfitting was partly avoided in the experiments by using the adopted evaluation methodology in section 4.2.1. In the adopted evaluation methodology, the development set of the SWAN dataset was used for early-stopping in the training of the neural networks to make sure the training and development sets are different in terms of variability in covariates as much as possible.

It has been shown that the low performance of machine learning models may be attributed to the *domain shift* present between datasets (V. M. Patel et al., 2015). Many domain adaptation and domain generalization methods have been developed to offer solutions to the domain shift problem (V. M. Patel et al., 2015; Wang and Deng, 2018). In the next chapters, I will investigate the low cross-dataset performance of CNN-based face PAD systems as a domain shift problem.

5 Domain Adaptation in PAD

As we saw in chapter 4, the performance of a PAD system may degrade significantly when tested in a *cross-dataset* scenario. This degradation can be attributed to the domain shifts present between datasets. The biometric sensor (camera device, in this case), the PAI used to create the attack, illumination, identity, pose of the subject, distance of the subject to the camera, and many other factors can cause domain shifts between datasets. Solutions to the problem of domain shift are called domain adaptation methods. Domain adaptation baselines and definitions were discussed in section 2.5.

In this chapter, two research hypotheses are investigated. First, in a neural network trained for PAD, it is hypothesized that only some layers are the main cause of performance degradation in cross-dataset evaluations due to domain shift and other layers are not. In other words, some layers are **domain specific** and are sensitive to domain shifts while other layers are **domain invariant**. Second, it is hypothesized that pruning some filters that are most sensitive to domain shift in the domain specific layers will improve the cross-dataset performance of PAD systems. Based on this hypothesis, a novel domain adaptation method is proposed that only uses *bona fide* samples from the target dataset for identifying filters that are most sensitive to domain shifts. The assumption behind the proposed approach is that collecting *bona fide* samples in the target domain is much less costly than collecting presentation attacks in the target domain. Moreover, in real-world scenarios, we may always expect a PAD system to be exposed to unseen attacks. Either the attack type may be different from those represented in the training set or the PAI used to create the attack may be different from those used during training. For example, for a model that is trained on print and replay attacks, a mask attack is an unseen attack type and an unknown paper and printer used to create a print attack is an attack with an unseen PAI.

To read this chapter, it is advised to read section 2.5 first which presents the related work on domain adaptation. Then, the hypothesis of domain specific layers is evaluated in section 5.1. The second hypothesis and the proposed domain adaptation method are presented in section 5.2. Extensive experiments are conducted and presented in sections 5.2.1 and 5.2.1.1. Finally, the results are discussed in section 5.3.

5.1 Which Layers in PAD CNNs are Domain Specific?

It has been shown by Zeiler and Fergus (2014) and Mallat (2016) that the initial layers (layers closer to input) in CNNs learn simple filters that extract **low-level** features. For example, the filters in initial layers of CNNs that are trained for object classification resemble filters such as Gabor filters, color blobs, and edge detectors. These *low-level* features are also called task independent features because they can be useful in a variety of tasks. Layers further down the network extract successively **higher-level** features, that correspond to complex and task-specific phenomena (Yosinski et al., 2014).

Furthermore, It has been shown in heterogeneous face recognition (Pereira, Anjos, and Marcel, 2019) and automatic speech recognition (Shor et al., 2019), that adapting only a few layers that are closest to input (layers responsible for *low-level features*) to a *target* dataset can significantly improve the performance on the target dataset. In other words, initial layers in those CNNs are domain specific and other layers are domain invariant (Pereira, Anjos, and Marcel, 2019).

In this section, it is hypothesized that only some layers in CNNs trained for PAD, are domain specific. That is, the highest performance of a CNN on a target dataset is achieved by adapting only the domain specific layers. To evaluate this hypothesis, first, the effect of domain shift on each layer in the DeepPixBiS architecture (see section 2.4.3 on page 38 for the details of the architecture) is investigated using feature divergence in section 5.1.1. Then, in section 5.1.2, several layers of the CNN are adapted to the target dataset one by one, to determine which layers provide the most benefit, when adapted to the target dataset. The benefit is estimated not only in terms of classification performance but also the number of parameters adapted. Finally, the results are discussed in section 5.1.3.

5.1.1 Feature Divergence

One way to quantify domain shift at each layer in a CNN is by computing **feature divergence** at each layer (X. Pan et al., 2018). This metric has been used in face PAD as well by Xiaoguang Tu, J. Zhao, et al., 2019. The definition of feature divergence is the following. Assume there exist two datasets that represents different domains, A and B. We want to determine, how often, on average, a specific filter in layer, L, is activated in each domain. Denote the average value of a filter over the spatial dimensions as f and assume a Gaussian distribution for f with mean μ and variance σ^2. Then the symmetric Kullback-Leibler (KL) divergence of this filter between domains A and B is:

$$D(f_A||f_B) = KL(f_A||f_B) + KL(f_B||f_A) \tag{5.1}$$

where

$$KL(f_A||f_B) = log\frac{\sigma_B}{\sigma_A} + \frac{\sigma_A^2 + (\mu_A - \mu_B)^2}{2\sigma_B^2} - \frac{1}{2} \tag{5.2}$$

Denote $D(f_{iA}||f_{iB})$ as the symmetric KL divergence of the ith filter in layer L. Then, the average feature divergence of layer L is given by:

$$D(L_A||L_B) = \frac{1}{C}\sum_{i=1}^{C} D(f_{iA}||f_{iB}) \tag{5.3}$$

where C is the total number of filters in layer L.

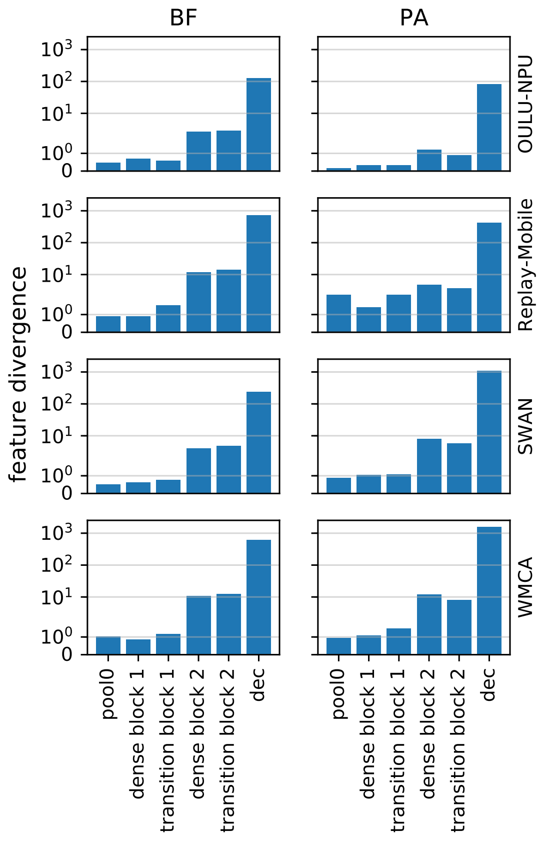


Figure 5.1 – Feature divergence of the DeepPixBiS architecture computed on four datasets. The DeepPixBiS architecture is trained on the *training* set of OULU-NPU and its feature divergence at each layer compared to the *test* sets of OULU-NPU, Replay-Mobile, SWAN, and WMCA are shown. The feature divergences are computed per layer and per class: *bona fide* (BF) and presentation attack (PA). The x axis indicates the layer name (see section 2.4.3 on page 38 for details of the DeepPixBiS architecture.). Layer *pool0* is closest to the input and layer *dec* is the (pixel-wise) logits layer. The y-axis is the divergence value calculated for each layer.

Feature divergence of each layer of the DeepPixBiS architecture between the *training* set of OULU-NPU and the *test* set of four datasets is shown in figure 5.1. The large feature divergence at the 'dec' layer is expected because it is the 'logits' layer where the network decisions are computed before applying the sigmoid activation and this is where the mistakes in classification appear. We can see that the last 3 final layers have the highest divergence compared to the first 3 initial layers. This indicates that the last 3 final layers are more sensitive to domain shifts. In the next section, we shall see how adapting these layers to the target domain can improve the performance of the network on the target domain.

5.1.2 Layer Adaptation

We observed the effect of domain shift on filter responses using feature divergence and saw that the last 3 layers had the highest divergence compared to other layers. Having a higher feature divergence at a layer (compared to other layers) may not be a good metric for telling us which layers to adapt to compensate for the domain shift present between the source and target datasets. In the following, we shall observe, through experimentation, that adapting which set of layers of the network to a target dataset will improve the performance of the network the most. The details of the adaptation experiments are given below.

The adaptation is done using the following approach in two stages:

1. A CNN is trained for PAD using the *training set* of the source dataset (here, OULU-NPU) and the cross entropy loss function. The training is stopped based on the accuracy on the *development set* of the source dataset.
2. The CNN is adapted using the same loss function and the *training set* of the target dataset. In this stage, only some layers may be adapted while keeping other layers fixed. The training is stopped based on the accuracy on the *development set* of the target dataset.

The new model is tested on the *test set* of the target dataset and the results are reported.

The DeepPixBiS architecture is used for experiments.[1] The architecture is trained on OULU-NPU and only some of its layers are adapted to our target datasets. Then, the performance of the model on the target dataset before and after adaptation is reported. The results for layer adaptation experiments are shown in figure 5.2. The *DeepPixBiS* system is considered the baseline and improvements of the adapted system are compared to this baseline. The performance is compared in terms of area under curve (AUC) of log-scale ROCs. Several combinations of layers are chosen for adaptation and the corresponding number of trainable parameters that are adapted is shown on the x-axis. The following results are observed in the figure:

- Adapting the first or the last layer of DeepPixBiS model does not always result in significant performance improvements.
- Adapting all layers significantly improves the performance in Replay-Mobile and SWAN datasets but not significantly in the WMCA dataset. The training data in the WMCA dataset is much smaller compared to the Replay-Mobile and SWAN datasets and adapting all parameters of the model on this small dataset leads to overfitting.

[1]All layers of the DeepPixBiS architecture, layers from *pool0* till *dec*, are convolutional layers. See section 2.4.3 on page 38 for more details on the architecture.

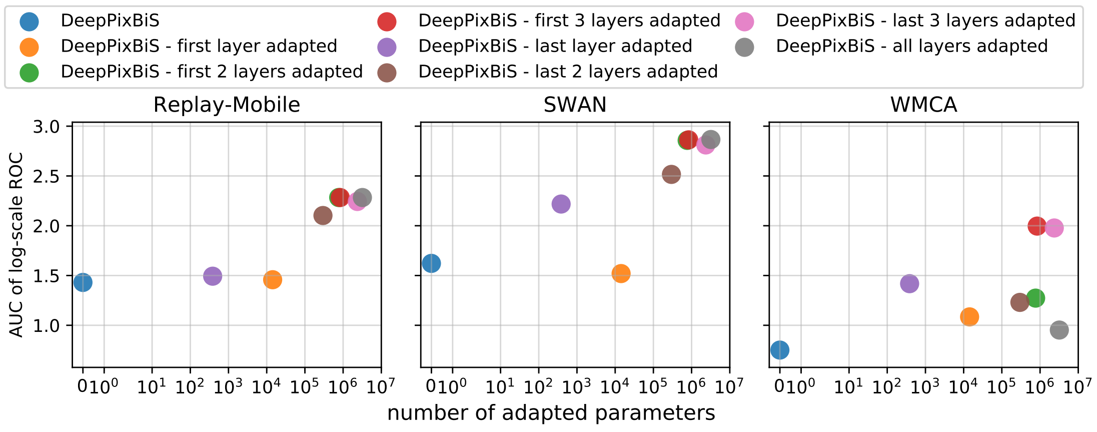


Figure 5.2 – Results of adapting different layers of the DeepPixBiS architecture on three target datasets. The DeepPixBiS model is trained on the *training set* of the OULU-NPU dataset. In each plot, the DeepPixBiS model is taken as the baseline and some of its initial or final layers are adapted on the target dataset using its *training* set and adapted performance of the model on the *test* set of the target dataset is reported. The performance of each model is reported as area under curve (AUC) of the log-scale ROC curves. The number of parameters that need to be adapted are shown on the x-axis. Results of adapting on Replay-Mobile, SWAN, and WMCA is reported. In the Replay-Mobile and SWAN plots, the dots related to *DeepPixBiS - first 2 layers adapted* and *DeepPixBiS - first 3 layers adapted* (green and red) methods overlap and the green dot is not easily visible in the plots. See section 2.4.3 on page 38 for details of the DeepPixBiS architecture.

- Adapting two layers (from beginning or end) of the network results in significant improvements over the baseline in all cases.
- Adapting the first or last three layers always result in most significant improvements in all three target datasets. Especially, when the target dataset is WMCA, adapting three layers results in significant improvements over the case when only two layers are being adapted.
- Overall, adapting the first 3 layers or the last 3 layers seems to result in a good ratio of performance improvement over the number of parameters that needs to be trained.

5.1.3 Discussion

To summarize, in a CNN trained for PAD, we observed that both the initial and final layers may be labeled as *domain specific* layers. That is, adapting either the initial or final layers improves the performance significantly on the target dataset. Adapting all layers may result in overfitting when limited training data form the target dataset is available. This overfitting was observed in the experiments when the CNN was adapted to the WMCA dataset. Following the notation used in section 2.5.3.2, if we assume that the first 3 layers of the DeepPixBiS architecture work as a *feature extractor*, M, and the last 3 layers work as a *classifier*, C, we may interpret the results in the following manner.

In the first stage, M_s and C_s, the feature extractor and the classifier, respectively, are trained using only the source-domain dataset. In the second stage, where only three layers of the network are adapted to the target-domain dataset. The second stage can be seen as either:

- training M_t and using it with C_s when adapting the first three layers only, or,
- training C_t and using it with M_s when adapting the last three layers only.

where M_t and C_t are feature extractor and classifier networks specific to the target domain, respectively. By training M_t and leaving C_s unchanged, a new feature extractor is trained that maps the samples from the target domain to the same distribution of features that C_s expects. This is similar to the approaches used in domain adaptation as we saw in section 2.5.3.

However, when training C_t while keeping M_s unchanged, a new classifier is designed with features constructed for the source domain. Some of these features may be relevant in the target domain and others may not. When C_s is adapted to learn C_t, the new classifier learns a different decision-boundary in the feature space that were constructed for the source domain. The new classifier may ignore the features that are not relevant in the target domain.

Learning C_t on the target domain and using it with M_s does not work in heterogeneous face recognition (HFR) (Pereira, Anjos, and Marcel, 2019) or automatic speech recognition (ASR) (Shor et al., 2019). However, in a network trained for PAD as a binary classification problem, adapting only the classifier part of the network does improve the performance on the target domain significantly. One reason that this approach does not work in HFR and ASR may be due to large domain shifts between source and target samples. For example, in HFR, one domain can be face images in the visual spectra and another domain can be thermal images of faces. This large differences between samples of source and target domains could render learned filters of M_s unusable in the target domain. Another reason may be that the classifier part of the networks on these tasks are much more complex compared to PAD. For example, HFR is an open set classification problem, i.e., the classes (identities) seen during training are different from classes at test time. This complex classifier may not be properly trained in the target domain where only a few classes (identities in FR) are available during training.

These differences suggest that, when we are correcting for domain shift in neural networks by adapting only a few layers, the choice of these layers depends on the task that the model is trained on. Based on these conclusions, I propose a novel domain adaptation approach in section 5.2 by pruning the filters in a CNN that are only relevant in the source domain.

5.2 Domain Guided Pruning of Neural Networks

As we saw in section 5.1, especially in figure 5.2, adapting the first three or last three layers of the DeepPixBiS architecture to the target-domain dataset results in significant performance improvements. For such domain adaptation method to be successful, sufficient training data from the target domain, for both *bona fide* and PA classes, is needed. However, it is not always feasible to collect data in the target domain. Notably, collecting PAs in the target domain may prove much more difficult and expensive, than collecting *bona fide* samples.

Here, it is hypothesized that, in a layer of a CNN trained for PAD, some of the learned filters are robust filters and generalize to the target domain and only some of the filters are specific to the source domain and do not generalize to the target domain. By pruning the filters that do not generalize to the target domain, it is assumed that the performance of the CNN will be improved on the target domain.

Based on this hypothesis, a novel domain adaptation method is proposed which identifies the filters that are most sensitive to domain shift in a layer and prunes them to improve the performance of the CNN on the target domain. Identifying these filters is done by using only *bona fide* samples of the target-domain dataset.

The details of the proposed method are as follows. Assume that there exist two datasets that represents different domains, A (source) and B (target), and the CNN model is trained on the source (A) domain:

1. Calculate the feature divergence (see equation (5.1)) of each filter F at layer L using only *bona fide* samples of the *training* sets of datasets A and B.
2. Prune N percent of the filters[2] of layer L which contribute to the most feature divergence values at layer L.
3. Then, re-train the layers $L+1$ and after on the *source* dataset (not the *target* dataset because it is assumed that no presentation attacks are available for training in the target dataset) using the same classification loss-function to account for the pruned filters.

The resulting model is evaluated on the *test* set of the target dataset.

Intuitively, this method works like a feature selection method. The first L layers following the input layer of the CNN may be seen as a *feature extractor*. Layers $L+1$ and after may be seen as a *classifier*. Then, by pruning *features* at layer L and retraining the classifier, the classifier is limited to use only robust features for prediction.

5.2.1 Experiments and Discussion

Implementation details of the proposed method are given in section 5.2.1.1 and the experiments are presented in section 5.2.1.2.

[2]Pruning can be done by multiplying the output of a filter by zero all the time or by removing the filter entirely from calculations to reduce the computation cost. Both methods result in the same behavior.

5.2.1.1 Implementation Details

The proposed *domain guided pruning* method is tested on two architectures: DeepPixBiS and MultiScale-InceptionResNetV2. See sections 2.4.3 and 4.1.6 on pages 38 and 87 for the details of the architectures, respectively. Layer L at which the filters are pruned is chosen as follows:

- **DeepPixBiS:** filters at layer *transition_block_1* (the third layer from the input) are pruned because adapting the last 3 layers (the layers after *transition_block_1*) showed the most significant improvement when adapting the network to a target dataset (see section 5.1).
- **MultiScale-InceptionResNetV2:** The layer L was chosen to be the layer *Concatenate* where the features from three different scales are concatenated. This layer is also the fourth layer from the end which makes this layer a similar choice as the chosen layer in DeepPixBiS.

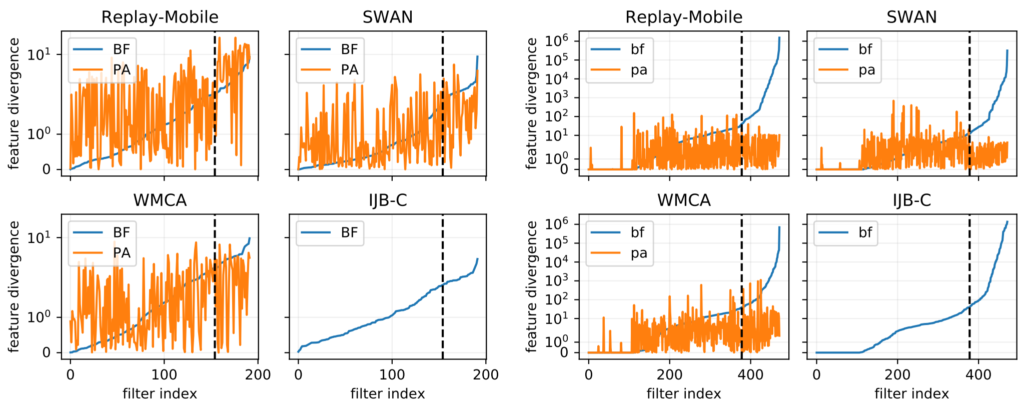
Both models are trained on the *training* set of OULU-NPU dataset and the training is stopped based on the accuracy on the *development* set of the same dataset. Feature divergences at layer L for each target dataset have been computed using *bona fide* samples of the corresponding training set, versus the training set of OULU-NPU. The N% of the filters that contributed most to the divergence values at layer L are pruned. The value for N was arbitrarily chosen to be 20. After pruning the model, layers $L+1$ and above were fine-tuned using the OULU-NPU dataset. The fine-tuning was also stopped based on the accuracy on the *development set* of OULU-NPU. The experiment results are discussed next.

5.2.1.2 Experiments

Because the domain guided pruning method uses only *bona fide* samples, a scenario is also tested where the model is not pruned using *bona fide* samples of the target dataset but rather, using a face recognition dataset. Still images of IARPA Janus Benchmark C (IJB-C)[3] face recognition dataset are used for this purpose. The low quality face images of this dataset were removed manually and around 3000 high quality face images were used for the experiments in this section.

The proposed method requires the feature divergences for each filter at layer L to be computed between the source and target datasets. I have computed the feature divergences for both *bona fide* and PAs. Although feature divergences for the PAs are not used for pruning, their values are computed and are shown here. This is useful to observe the feature divergences of PAs for filters that are pruned and for the ones that are not pruned. The feature divergences between OULU-NPU and 4 datasets are shown in figure 5.3. Results for the layer *transition_block_1* in the *DeepPixBiS* architecture are shown in figure 5.3a. Results for the layer *Concatenate* in the *MultiScale-InceptionResNetV2* architecture are shown in figure 5.3b. In each plot, the filter indexes are sorted in ascending order of divergence values calculated using the *bona*

[3]https://www.nist.gov/programs-projects/face-challenges



(a) – DeepPixBiS **(b)** – MultiScale-InceptionResNetV2

Figure 5.3 – Feature divergence values of each filter at layer L in two CNNs (DeepPixBiS and MultiScale-InceptionResNetV2) are shown. The results for the *transition_block_1* layer in the DeepPixBiS architecture and the *Concatenate* layer in the MultiScale-InceptionResNetV2 architecture are presented. The feature divergences are calculated separately for *bona fide* (BF) and presentation attack (PA) samples between the source dataset (OULU-NPU) and the target datasets: Replay-Mobile, SWAN, WMCA, and IJB-C. In each subplot, filters are sorted based on their feature divergences using *bona fide* samples. The index at which $N = 20\%$ of the filters will be pruned is shown using vertical dashed black lines. All filters at the right of this index will be pruned. *Training sets* of the datasets are used for these calculations.

fide samples. The index at which $N = 20\%$ of the filters would be pruned is shown using a vertical dashed black line. Because there are no presentation attacks in the FR IJB-C dataset, only divergences using the *bona fide* samples are shown. Although the filters are being pruned solely based on their divergence values computed using only *bona fide* samples, we may observe in the figures that some of these filters also have high divergence values when tested on presentation attacks as well.

The results for the proposed domain guided pruning method applied to the DeepPixBiS architecture are shown in figure 5.4. As detailed in section 5.2.1.1, in each experiment, the CNN is first trained on OULU-NPU, pruned using *bona fide* samples of a target dataset, fine-tuned again on OULU-NPU, finally evaluated on all PAD datasets. The dataset that was used for pruning is mentioned in the legends of the figure. The datasets that were used for pruning were: IJB-C, Replay-Mobile, SWAN, and WMCA. We can draw several conclusions form this figure:

1. We can observe that pruning does not significantly affect the performance of the model on the *source* dataset. This is shown in the results by testing all models on the OULU-NPU dataset.

2. We can observe the performance improvement between the baseline and the proposed domain guided pruning method when the both the pruning and test datasets are the same. For example, comparing the performance of the original DeepPixBiS model on the Replay-Mobile dataset with that of the new version of the DeepPixBiS model pruned

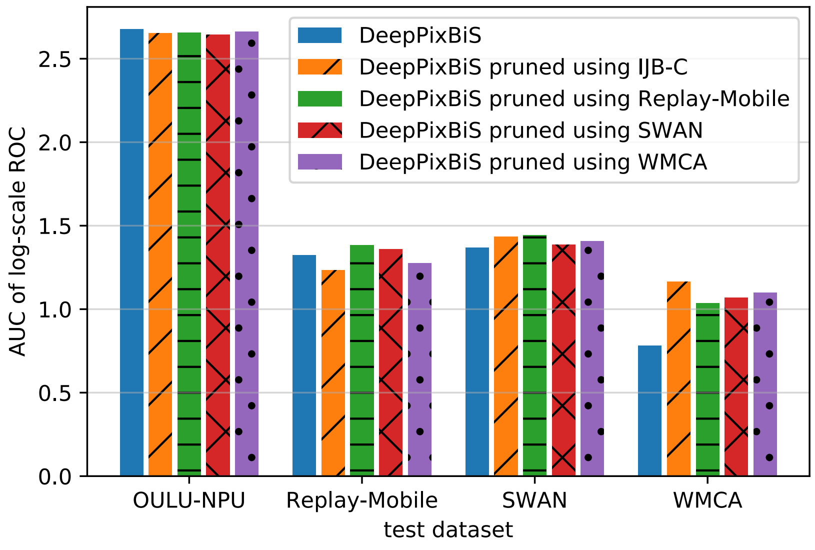


Figure 5.4 – Results of the domain guided pruning experiments using the DeepPixBiS architecture. The area under the curve (AUC) values of log-scale ROC are shown. The higher the value the better is the performance of the system. All models are both trained and fine-tuned after pruning on OULU-NPU. The dataset used for pruning is mentioned in the legend. The dataset that the model was tested on is shown on the x axis. The models are compared to the baseline when no pruning is performed.

using the Replay-Mobile dataset, we note that the new model performs significantly better on the target (Replay-Mobile) dataset. We can see that pruning using the target dataset improves the performance of the model when tested on the target dataset. This improvement is clearly visible in the case of the WMCA dataset where AUC is increased from ~ 0.8 to ~ 1.1.

3. The performance of the model pruned using a different dataset than the test dataset can also be observed. For example, we can see that when the DeepPixBiS architecture is pruned using the IJB-C dataset, its performance degrades slightly on the Replay-Mobile dataset, improves slightly on the SWAN dataset, and improves significantly on the WMCA dataset.

The results for the proposed domain guided pruning method for the MultiScale-InceptionResNetV2 are shown in figure 5.5. As with the experiments for the DeepPixBiS model, all models are trained on the *training* set of OULU-NPU, pruned using *bona fide* samples of a target dataset, fine-tuned again on OULU-NPU, finally, evaluated on all PAD datasets. The dataset used for pruning is mentioned in the legends of the figure. Again, in this experiment, the following datasets have been used for pruning: IJB-C, Replay-Mobile, SWAN, and WMCA. From the figure, we can make the following observations:

1. We can observe that pruning improves the performance of the model significantly on

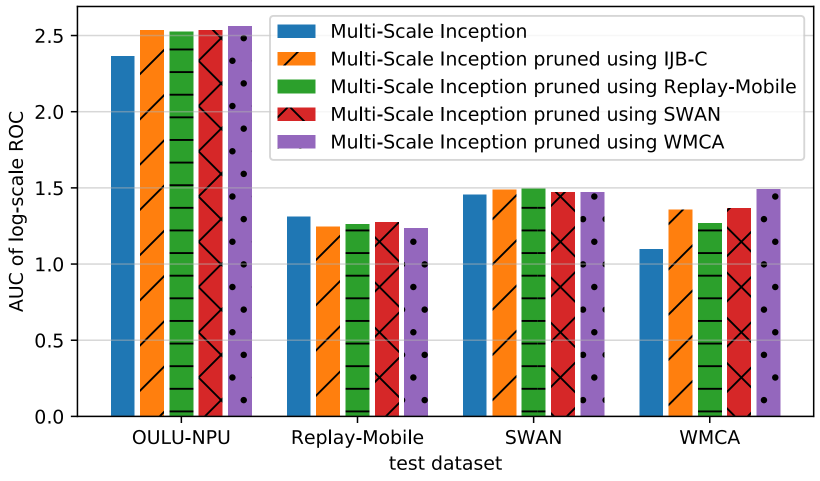


Figure 5.5 – Results of the domain guided pruning experiments using the MultiScale-InceptionResNetV2 architecture. The area under the curve (AUC) values of log-scale ROC are shown. The higher the value the better is the performance of the system. All models are both trained and fine-tuned after pruning on OULU-NPU. The dataset used for pruning is mentioned in the legend. The dataset that the model was tested on is shown on the x axis. The models are compared to the baseline when no pruning is performed.

the *source* dataset. This is shown in the results by testing all models on the OULU-NPU dataset.

2. We can observe the performance improvement between the baseline and the proposed domain guided pruning method when the both the pruning and test datasets are the same. For example, comparing the performance of the original MultiScale-InceptionResNetV2 model on the WMCA dataset with that of the new version of the MultiScale-InceptionResNetV2 model pruned using the WMCA dataset, we note that the new model performs significantly better on the target (WMCA) dataset. We can see that pruning using the target dataset either does not change the performance significantly or improves the performance significantly when tested on the target dataset. The performance of the model degrades slightly when the target dataset is Replay-Mobile, increases slightly when the target dataset is SWAN, and improves significantly when the target dataset is WMCA. In the case of the WMCA dataset, AUC is increased from ~ 1.1 to ~ 1.5.

3. The performance of the model pruned using a different dataset than the test dataset can also be observed. For example, we can see that when the MultiScale-InceptionResNetV2 architecture is pruned using the IJB-C dataset, its performance degrades slightly on the Replay-Mobile dataset, improves slightly on the SWAN dataset, and improves significantly on the WMCA dataset.

5.3 Conclusions

In this chapter, the generalization problem of PAD systems was formulated as a domain adaptation problem. In section 5.1, it was shown that when only a small amount of data from the target domain is available, as is the case with the WMCA dataset, the best strategy for adaptation can be adapting only a few layers of the CNN model to the target domain instead of adapting all layers. For example, adapting the last 3 layers of the DeepPixBiS architecture to the WMCA dataset resulted in AUC improving from ~ 0.8 to ~ 2.0 while adapting all layers of the architecture resulted in an AUC of ~ 0.95. Contrary to other works (Pereira, Anjos, and Marcel, 2019; Shor et al., 2019), the experiments presented in this chapter indicate that both initial as well as final layers of a CNN model trained for PAD may show domain-specific behavior. In other words, the notion of *domain specific* and *domain invariant* layers varies depending on the task.

Adaptation of the layers was done based on the assumption that enough training data from both classes of *bona fide* and presentation attacks from the target domain were available. For cases where only *bona fide* data is available from the target dataset, the *domain guided pruning* method was proposed in section 5.2. I argue that collecting new *bona fide* samples in a target domain can be significantly less costly than collecting presentation attacks. Also, we can never collect data for *unseen* attacks (different attack types and PAIs used in attacks) in the target domain.

The proposed domain guided pruning method was tested on three target datasets: Replay-Mobile, SWAN, and WMCA and two CNN architectures: DeepPixBiS and MultiScale-InceptionResNetV2. We showed that:

- when the model was pruned using the target dataset, the performance of the model increased on the target dataset. This was true in 5 out of 6 cases. Compared to other test datasets, the baseline models were performing worse on the WMCA dataset. Pruning significantly improved the performance of the models on the WMCA dataset. In case of DeepPixBiS and the WMCA dataset, AUC improved from ~ 0.8 to ~ 1.1. Similarly, in case of MultiScale-InceptionResNetV2 and the WMCA dataset, AUC improved from ~ 1.1 to ~ 1.5.
- pruning either did not degrade the performance of the model on the source dataset, when the model was DeepPixBiS, or improved it when the model was MultiScale-InceptionResNetV2. In the case of MultiScale-InceptionResNetV2, the AUC improved from ~ 2.3 to ~ 2.5.

These results give us confidence that the proposed pruning method is applicable to intra-dataset and cross-dataset evaluation scenarios.

In section 5.2, we also showed that the domain guided pruning method can also be implemented as a *domain generalization* method. This was done by pruning the models using *bona* *fide* samples of a face recognition (FR) dataset and testing the models on unseen PAD datasets. This approach slightly degraded the performance on Replay-Mobile, slightly improved the performance on SWAN, and significantly improved the performance on the WMCA dataset. In case of DeepPixBiS and the WMCA dataset, AUC improved from ~ 0.8 to ~ 1.2. Similarly, in case of MultiScale-InceptionResNetV2 and the WMCA dataset, AUC improved from ~ 1.1 to ~ 1.4. This indicates that a FR dataset, being a large dataset with a large variety of several factors (such as identities, pose, illumination, and camera device), may fully represent some of the factors that are only partially represented in PAD datasets. In other words, for some of the factors that are different between PAD datasets, a FR dataset may represent all possible variations of those factors.

In the next chapter, we shall see other approaches of using FR datasets to improve the performance of PAD models. Several works on multi-task learning where both a FR dataset and a PAD dataset are used to train a multi-task model (the tasks being FR and PAD) will be presented. Then, a novel unsupervised *domain generalization* method will be presented which uses FR datasets and autoencoders.

6 Domain Generalization in PAD

In chapter 4, we observed that the performance of the PAD systems drops significantly when evaluated under cross-dataset scenarios compared to intra-dataset evaluations. As discussed in chapter 5, the low cross-dataset performance of PAD systems can be attributed to domain shift. Several domain adaptation methods have been proposed to compensate for domain shift (Csurka, 2017), see section 2.5 on page 40. These methods rely on limited training data from the target domain. In the context of biometrics, often the target domain is unknown (Marcel et al., 2019). In such cases, it is impossible to collect data in the target domain. When the target domain is known in advance, collecting *bona fide* samples is less expensive than collecting PAs. Therefore, in chapter 5 I proposed a novel domain adaptation method that uses only *bona fide* samples from the target domain to compensate for domain shift.

On the other hand, domain generalization methods do not rely on training data from the target domain. Instead, they use data from *multiple* source domains to train models that are invariant to domain shifts. We assume that if a model is invariant to domain shifts present between multiple source domains, it will also generalize to other unseen (target) domains. As discussed before, both MMD- and adversarial-based domain adaptation methods (see section 2.5.3 on page 42) can be implemented as domain generalization methods. These domain adaptation methods match the distribution of the source and target samples in a *learned feature space*. They assume that if the source and target domain features have the same distribution, then a classifier that is trained on the features of the source domain can be used to classify samples from the target domain. To implement these methods as domain generalization methods, all that is required is to match the distribution of the multiple source domains in the learned feature space instead. Therefore, data from *multiple known domains* is needed when using domain generalization methods.

However, collecting data in multiple domains can prove to be difficult as well. To collect data in multiple domains, we must first identify the **nuisance factors** (also known as **nuisance variables** or **nuisance parameters**) where variations of these factors is the cause of domain shift between datasets. An ideal PAD system would be invariant to nuisance factors. In face PAD datasets, domain-shift may be caused by a variety of nuisance factors, including: the camera device, resolution of images, distance of the subject to the camera, the instrument used to create the attack (PAI), lighting conditions, identity, pose, age, and facial-makeup. Identifying all the nuisance factors is impossible (Y. Bengio, Courville, and Vincent, 2013). Some factors may be known such as the camera device and the lighting conditions but we may not be aware of all the factors. Moreover, categorizations of factors is also difficult if not impossible (Y. Bengio, Courville, and Vincent, 2013). For example, to change the camera device between datasets, we might only consider cameras in mobile phones and professional cameras as two different domains, or we might consider each camera model as a different domain. Also, categorization of factors may be subjective. For example, defining what different variations of lighting conditions exist is subjective.

There are many methods that induce invariance to nuisance factors in a learned model (Y. Bengio, Courville, and Vincent, 2013; Xie et al., 2017; Louizos et al., 2015; Ganin et al., 2016; Y. Li, Swersky, and Zemel, 2014). However, most of these methods require the nuisance factors to be known and labeled in the training data. One recent method that proposes to induce invariance to all nuisance factors in an unsupervised manner is *unsupervised adversarial invariance (UAI)* (Jaiswal et al., 2018) which will be detailed in section 6.1. UAI attempts to identify and separate nuisance factors from other underlying factors of data. The learned model would classify samples without taking nuisance factors into account. However, there is actually no guarantee that the method is separating all the nuisance factors from other factors. In fact, this method is very susceptible to biases in the training data which I will explain in section 6.1. In (Jaiswal et al., 2019), where the authors use UAI for face PAD, no cross-dataset performance evaluation is reported while reporting cross-dataset performance of face PAD systems is commonplace (George and Marcel, 2019; Xiaoguang Tu, J. Zhao, et al., 2019; Xiaoguang Tu, H. Zhang, et al., 2019).

Moreover, current face PAD datasets (see section 2.7 on page 59) do not contain enough variability of nuisance factors so that *data-driven* PAD methods can model the nuisance factors accurately. For example, face PAD datasets are usually collected with less than 10 camera devices, with only 50 to 150 identities participating, and have limited variations in lighting conditions. FR datasets, on the other hand, contain millions of face images with hundreds of thousands of identities which are adequately varied (Guo et al., 2016). Being such large and varied datasets, they can be used to adequately model some nuisance factors of face PAD systems.

In this chapter, I hypothesize that *all* the underlying factors that explain the samples of an FR dataset (which contains only *bona fide* samples) are nuisance factors in a face PAD system. For example, some of these factors are the camera device, the lighting condition, and the identity of the person. All these factors exist in both *bona fide* and PA face images and are nuisance factors in PAD. Based on this hypothesis, I propose a novel method which induces invariance to these factors in a face PAD system by explicitly modeling these factors in an unsupervised manner using an FR dataset.

To the best of my knowledge, this is the first work to take advantage of FR datasets to improve generalization of face PAD systems in an *unsupervised* manner. Other works such as (Xiaoguang Tu, J. Zhao, et al., 2019; Ying, X. Li, and Chuah, 2018) use multi-task learning of both FR and PAD which require the face images to be labeled according to identities in both FR and PAD datasets. However, identity labels are not strictly necessary in PAD datasets and some PAD datasets may not have them. Moreover, these methods only use an FR dataset for initialization of a multi-task network. That is, the FR part of the network is pre-trained on a large FR dataset. Then, the network is trained on a small PAD dataset with a few identities for both tasks of PAD and FR.

This chapter is organized in the following way. Previous works related to the proposed method are presented in section 6.1. The proposed method is detailed in section 6.2. Implementation details are outlined in section 6.3. Experiments are described in section 6.4, and section 6.5 contains experimental analysis to further support the claims. Finally, conclusions are made in section 6.6.

6.1 Related Work

Two related works that induce invariance to nuisance factors are detailed below.

6.1.1 Unsupervised Adversarial Invariance (UAI)

One recent method that proposes to induce invariance to all nuisance factors in an unsupervised manner is **unsupervised adversarial invariance (UAI)** (Jaiswal et al., 2018). In UAI, a neural network is trained simultaneously for two tasks: the required primary task (classification or regression), and reconstruction of the input. After a few initial layers, the network splits into two branches, each dedicated to optimizing one task. The initial layers produce two embeddings: e_1 and e_2 which will be used in the two task-specific branches. The reconstruction branch takes two inputs: e_2, and $\hat{e}_1$, a noisy version of e_1; e_1 is multiplied by a random Bernoulli noise to construct $\hat{e}_1$. The input to the branch responsible for the primary task is only e_1. Two adversarial losses are added which make sure e_1 and e_2 do not contain duplicate information (see (Jaiswal et al., 2018)). For reconstruction, most factors of the data are needed to correctly reconstruct the input. Since reconstruction is done using e_2 and $\hat{e}_1$ (which is noisy), it is assumed that most factors of the data will be represented by e_2 to guarantee correct reconstruction of input. Also, because by construction e_1 and e_2 do not contain duplicate information, only factors crucial for the primary task will be represented by e_1. However, there is not guarantee that e_1 will not represent any nuisance factor. In fact, if the dataset contains a bias that may simplify the primary task, there is no guarantee that it will not be represented in e_1. Thus, e_1 could represent the bias of the dataset. Moreover, in this method, the nuisance factors are modeled using the dataset for the primary task. In case of face PAD, these datasets may not fully represent all the possible nuisance factors. In (Jaiswal et al., 2019), where the authors use UAI for face PAD, no cross-dataset performance evaluation is reported.

6.1.2 Inter-Session Variability (ISV)

The inter-session variability (ISV) technique (which was detailed in section 2.3.3) proposed by Vogt and Sridharan (2008) and Wallace et al. (2011) explicitly models within-class variations (nuisance factors) in Gaussian Mixture Model (GMM)-based biometric recognition systems (Reynolds, Quatieri, and Dunn, 2000; Cardinaux, Sanderson, and S. Bengio, 2006). They assume that samples from all classes (identities in FR) have the same nuisance factors and that these factors are contained in a linear subspace of GMM mean supervector space. Then, a training mechanism is proposed to explicitly model these nuisance factors in the GMM mean supervector space. Once these factors are modeled, given a face image and its GMM-based mean supervector, its nuisance factors are estimated and their effect is removed from the mean supervector. The obtained mean supervector is used for classification instead of the original mean supervector. While this method has been successfully applied on FR, it is limited to GMM-based systems which use hand-crafted features. Since then, many deep learning based FR algorithms have outperformed GMM-based methods in FR (Sandberg, 2017; Parkhi, Vedaldi, and Zisserman, 2015; X. Wu et al., 2015).

6.2 Proposed Method

Many nuisance factors can cause domain shifts in face PAD datasets. In this work, we assume that all factors present in *bona fide* face images are also present in PA face images. For example, factors such as identities, lighting conditions, and camera devices can be different in both *bona fide* and PA samples between datasets. Here, we propose a method to explicitly model these common factors using an FR dataset. We assume that these factors are well represented in an FR dataset which contains millions of *bona fide* face images. By explicitly modeling these factors, we can induce invariance to these factors in a PAD system.

More specifically, assume that each face image, $\mathbf{I}$, is generated through a function, f, and some noise, ϵ:

$$\mathbf{I} = \mathrm{f}(\mathbf{y}, \mathbf{z}_1, \mathbf{z}_2) + \epsilon \tag{6.1}$$

where $\mathbf{y}$ is the variable that we want to predict – whether $\mathbf{I}$ is a PA, $\mathbf{z}_1$ and $\mathbf{z}_2$ are multivariate latent variables. The variable $\mathbf{z}_1$ represents all the nuisance factors present in *bona fide* samples that are present in PA samples as well, whereas $\mathbf{z}_2$ represents all other nuisance factors that are exclusive to PAs. The variable $\mathbf{z}_1$ may encapsulate information about gender, pose, identities, lighting condition, camera characteristics, and so on. The variable $\mathbf{z}_2$ may contain information about the presentation attack instrument (PAI). In this work, we will not model $\mathbf{z}_2$ or try to induce invariance to $\mathbf{z}_2$. However, if some factors present in $\mathbf{z}_2$ are known and labeled, it is possible to induce invariance to these factors using traditional invariance induction methods such as (Ganin et al., 2016).

Assume that f can be modeled as the sum of two other functions:

$$\mathrm{f}(\mathbf{y},\mathbf{z}_1,\mathbf{z}_2) = \mathrm{g}(\mathbf{z}_1) + \mathrm{h}(\mathbf{y},\mathbf{z}_2) \tag{6.2}$$

where the functions g and h each produce an image given their respective latent variables as input. The final image is the sum of these two images plus some noise. Given an image $\mathbf{I}$, assume that $\mathbf{z}_1$ can be estimated through some function, e:

$$\mathbf{z}_1 = \mathrm{e}(\mathbf{I}) \tag{6.3}$$

and that the function g is also known. This allows us to reconstruct a face image using only $\mathbf{z}_1$:

$$\mathbf{I}_{z_1} = \mathrm{g}(\mathbf{z}_1) = \mathrm{g}(\mathrm{e}(\mathbf{I})) \tag{6.4}$$

Then, given a *bona fide* or PA face image, we can approximate the output of h as:

$$\mathrm{h}(\mathbf{y},\mathbf{z}_2) \approxeq \mathbf{I} - \mathbf{I}_{z_1} = \mathbf{I}_{y,z_2} \tag{6.5}$$

where $\mathbf{I}_{y,z_2}$ is the **reconstruction-error image** which is the error between an image and its reconstruction using e and g. Because $\mathbf{I}_{y,z_2}$ is not influenced by the factors related to $\mathbf{z}_1$, it can be used instead of $\mathbf{I}$ to train a PAD system.

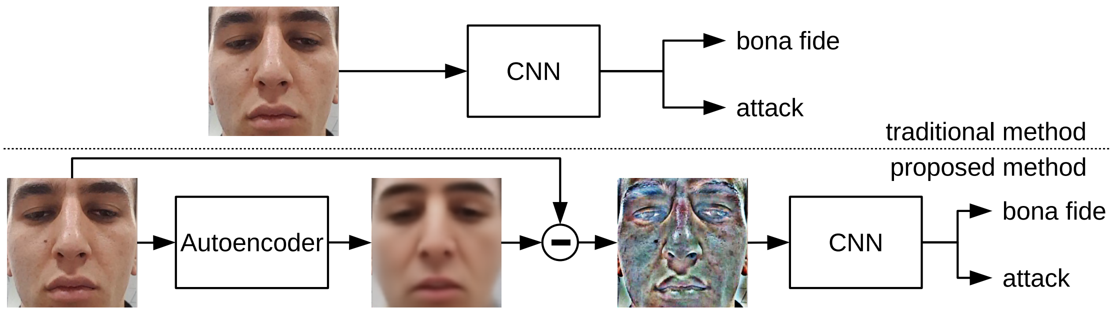


Figure 6.1 – Diagram of the proposed method. The upper part depicts the traditional approach of training a PAD system, where the original face images are used to train a CNN PAD system. In the proposed method, shown in the lower part of the figure, the autoencoder is first trained to reconstruct faces, using a large FR dataset. Then the reconstruction-error images computed from the output of this autoencoder are used to train the (CNN) PAD system. The autoencoder is not updated when the PAD-CNN is trained. The CNN is trained on reconstruction-error images of a PAD dataset.

Functions e and g can be modeled in an unsupervised manner using an *information maximizing variational autoencoder* (Info-VAE) (S. Zhao, Song, and Ermon, 2017) (which was detailed in section 2.2.9). The encoder and decoder parts of the autoencoder approximate e and g, respectively. Info-VAEs are able to learn *meaningful and disentangled* representations of samples where each dimension in the learned representation can represent one underlying factor of the data (S. Zhao, Song, and Ermon, 2017). Info-VAEs learn these representations by imposing a prior distribution on their latent variables. By training an Info-VAE to reconstruct face images using only *bona fide* samples of an FR dataset, the autoencoder will model $\mathbf{z}_1$ as its latent variable. The trained autoencoder, when tested against *bona fide* and PA face images of a PAD dataset, will reconstruct face images only in terms of factors that it has modeled. In other words, the encoder, e, encodes any face image to its learned factors, $\mathbf{z}_1$, and the decoder reconstructs the face image using only those factors. The diagram of the proposed method is shown in figure 6.1. The proposed method adds a pre-processing step using a pre-trained autoencoder to the PAD system compared to traditional methods. Instead of using original face images as input to a PAD system, we use the reconstruction error image of the autoencoder. Some examples of reconstruction error images are shown in figure 6.2. We may observe that reconstruction error images look more similar to each other compared to the original images; The reconstruction error images are similar to each other in terms of color, contrast and so on. This is due to the removal of some of the nuisance factors.

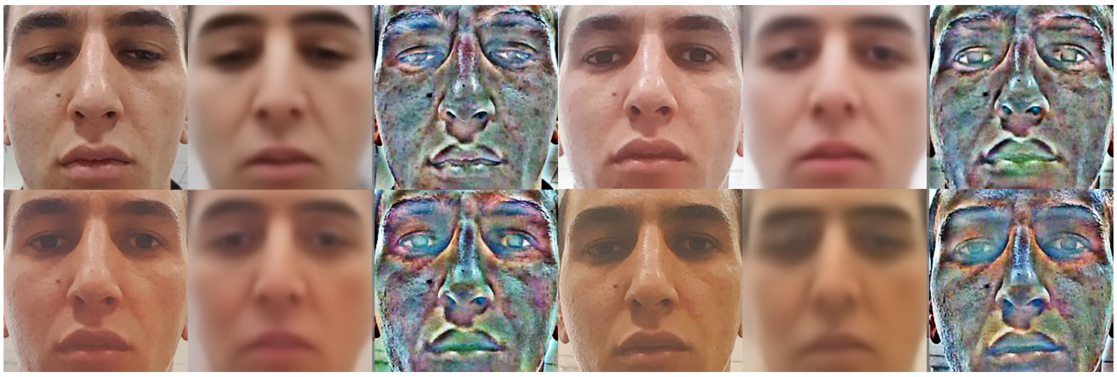


Figure 6.2 – Examples of autoencoder reconstruction-error images. The images in each three columns, from left to right, are original images, reconstructed images by the autoencoder, and reconstruction error images. The original image in top left is a *bona fide* sample and the rest of original images are PAs. The reconstruction error images contain less variations compared to the original images.

6.3 Implementation Details

The following face PAD datasets that have been used in this study: OULU-NPU (Zinelabinde Boulkenafet et al., 2017), Replay-Mobile (Costa-Pazo et al., 2016), SWAN (Ramachandra et al., 2019), and WMCA (George et al., 2019). See section 2.7 on page 59 for details on the datasets.

For the experiments discussed in section 6.4, all models have been trained on OULU-NPU, using the evaluation methodology described in section 4.2.1, and evaluated on all four PAD datasets. The classification performance is reported in terms of area-under-the-curve (AUC) of *log-scale* receiver operating characteristic (ROC) curves. The ROC curves are computed with $APCER$ along the x-axis (log-scale) and $1 - BPCER$ on the y-axis[1]. See section 2.6 on page 48 for details on performance evaluation.

[1]Note that because AUC of *log-scale* ROCs are reported, their values can be higher than 1.

The proposed method is tested against the DeepPixBiS CNN architecture (George and Marcel, 2019) which is detailed in section 2.4.3. The architecture for the encoder part of the autoencoder is a DenseNet-161 (G. Huang et al., 2017) (see section 2.2.8 on page 21) and the architecture of the decoder is a slightly modified version of the face generator in (Miyato, Kataoka, et al., 2018) which is detailed in table 6.1. The size of $\mathbf{z}_1$, the latent variable of the autoencoder, is arbitrarily chosen to be 256 and its prior is arbitrarily assumed to be a Gaussian distribution with mean 0 and standard deviation of 3 (diagonal covariance matrix). The autoencoder is trained using cleaned versions of Microsoft Celeb (MS-Celeb-1M) (Guo et al., 2016) and the Celeb-A (Z. Liu et al., 2015) FR datasets jointly.

Table 6.1 – Architecture details of the decoder part of the autoencoder used in the experiments. The input to the architecture is a 256 dimensional vector which is the latent variable. See section 2.1 on page 9 for more details on CNNs.

Layer (type)	Details	Output Shape	Number of Parameters
reshape (Reshape)		(1, 1, 256)	0
dconv_0 (Conv2DTranspose)	filters=512, kernel_size=(7, 7), strides=(7, 7)	(7, 7, 512)	6,422,528
crop_0 (Cropping2D)	cropping=((0, 0), (0, 0))	(7, 7, 512)	0
bn_0 (BatchNormalization)		(7, 7, 512)	1,536
relu_0 (Activation)	activation=relu	(7, 7, 512)	0
dconv_1 (Conv2DTranspose)	filters=256, kernel_size=(4, 4), strides=(2, 2)	(16, 16, 256)	2,097,152
crop_1 (Cropping2D)	cropping=((1, 1), (1, 1))	(14, 14, 256)	0
bn_1 (BatchNormalization)		(14, 14, 256)	768
relu_1 (Activation)	activation=relu	(14, 14, 256)	0
dconv_2 (Conv2DTranspose)	filters=128, kernel_size=(4, 4), strides=(2, 2)	(30, 30, 128)	524,288
crop_2 (Cropping2D)	cropping=((1, 1), (1, 1))	(28, 28, 128)	0
bn_2 (BatchNormalization)		(28, 28, 128)	384
relu_2 (Activation)	activation=relu	(28, 28, 128)	0
dconv_3 (Conv2DTranspose)	filters=64, kernel_size=(4, 4), strides=(2, 2)	(58, 58, 64)	131,072
crop_3 (Cropping2D)	cropping=((1, 1), (1, 1))	(56, 56, 64)	0
bn_3 (BatchNormalization)		(56, 56, 64)	192
relu_3 (Activation)	activation=relu	(56, 56, 64)	0
dconv_4 (Conv2DTranspose)	filters=32, kernel_size=(4, 4), strides=(2, 2)	(114, 114, 32)	32,768
crop_4 (Cropping2D)	cropping=((1, 1), (1, 1))	(112, 112, 32)	0
bn_4 (BatchNormalization)		(112, 112, 32)	96
relu_4 (Activation)	activation=relu	(112, 112, 32)	0
dconv_5 (Conv2DTranspose)	filters=16, kernel_size=(4, 4), strides=(2, 2)	(226, 226, 16)	8,192
crop_5 (Cropping2D)	cropping=((1, 1), (1, 1))	(224, 224, 16)	0
bn_5 (BatchNormalization)		(224, 224, 16)	48
relu_5 (Activation)	activation=relu	(224, 224, 16)	0
dconv_6 (Conv2DTranspose)	filters=3, kernel_size=(1, 1), strides=(1, 1)	(224, 224, 3)	51
crop_6 (Cropping2D)	cropping=((0, 0), (0, 0))	(224, 224, 3)	0
tanh_6 (Activation)	activation=tanh	(224, 224, 3)	0
Model	Parameters: total=9,219,075, trainable=9,217,059		

The reconstructed images generated by the autoencoder resemble low-pass filtered versions of the original images. Consequently, the reconstruction-error images will mainly contain the high frequency information of the original image. However, this approach is different from directly extracting high frequency components of the image based on a Gaussian-blur filter. To show the difference, the proposed method is also compared with a PAD system that is trained on difference images between blurred images and original images. This system will be called *Blur Error* in the experiments. In total, we will compare four methods:

1. *DeepPixBiS* (baseline): our baseline CNN.
2. *Blur Error* (BE) (baseline): similar to the baseline but the input image $\mathbf{I}$ is first blurred using a Gaussian kernel, and the difference-image ($\mathbf{I} - \mathbf{I_{blurred}}$) is used to train the baseline CNN.
3. *Autoencoder Error* (AE) (proposed): like the baseline but the input images to DeepPixBiS are the reconstruction-error images of a pre-trained autoencoder.
4. *Thresholded Autoencoder Error* (TAE) (proposed): similar to *AE* and is detailed below.

In the *TAE* method, input images that do not meet certain quality criteria are rejected (not processed or scored) by the PAD system. In preliminary experiments I observed that the autoencoder cannot adequately reconstruct certain face images, such as very dark faces or faces with extremely non-frontal poses. Since the prior distribution for $\mathbf{z}_1$ is known, we can use the likelihood of each sample as a quality metric to reject unusual input images. If the likelihood of a sample is too small, the autoencoder is not able reconstruct the face image correctly because the decoder has not seen $\mathbf{z}_1$ values outside of the prior distribution. Figure 6.3 shows some face-image examples, the corresponding reconstructions and log-likelihood values. In our experiments we have set the threshold for the log-likelihood at -600 for rejecting samples. This threshold has been selected based on manual inspection of results in preliminary experiments. Overall, after thresholding the face images and rejecting some frames in videos, between 5% to 19% of videos were rejected, depending on the test dataset.

6.4 Experiments

Results of evaluating the various networks on different datasets are shown in figure 6.4. The various datasets are shown on the horizontal-axis. Results for the OULU-NPU dataset correspond ot the intra-dataset evaluations. The best performing method is the *DeepPixBiS* baseline in intra-dataset evaluations. The performance of the BE method is slightly lower than that of *DeepPixBiS*, and the performance of the proposed AE and TAE methods are even worse. However, we can argue that the baseline methods are overfitting on the OULU-NPU dataset in intra-dataset evaluations as their performance degrades significantly in the cross-dataset scenario. Cross-dataset evaluations can be seen when the test dataset is not OULU-NPU. Overall, we can see that the TAE method performs slightly better than the AE method in all cross-dataset evaluations. For the SWAN and the WMCA datasets, both proposed methods (AE and TAE) perform significantly better compared to the baselines. For the Replay-Mobile dataset, however, all methods show similar performance, and the proposed AE and TAE methods perform slightly worse compared to the baselines.

I have investigated the low performance of proposed methods in the case of Replay-Mobile. In this dataset, some face images are either very dark or have very strong lateral illumination. These samples are annotated with lighting conditions of *adverse* and *lateral* in the dataset.

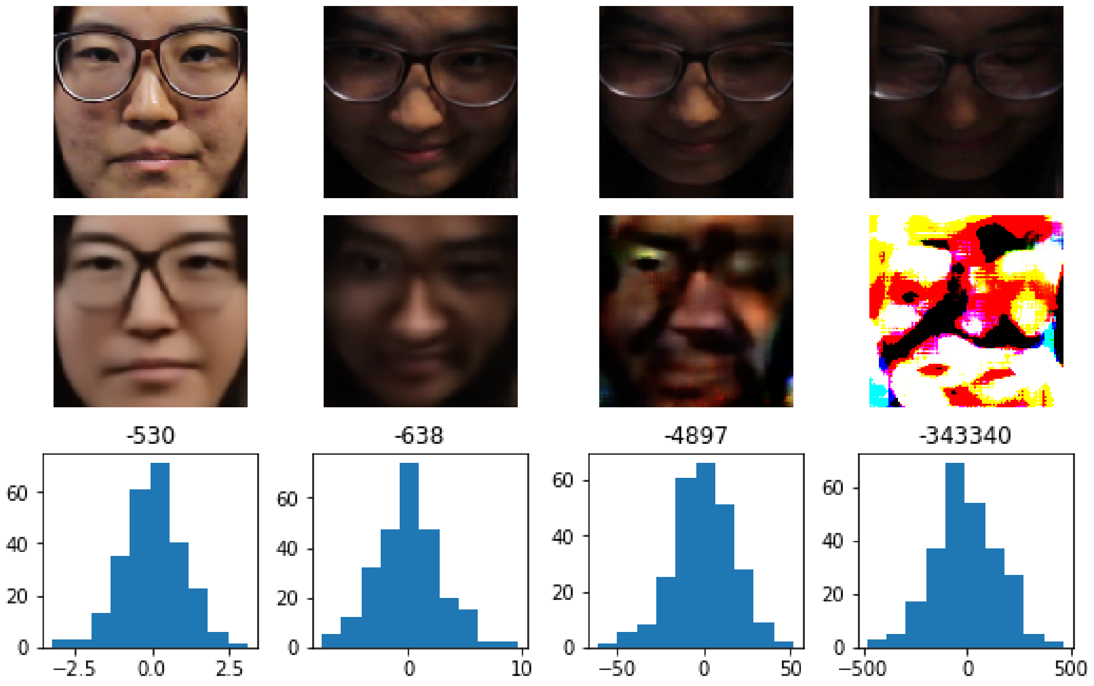


Figure 6.3 – Examples of the reconstructions of the autoencoder. Each row, from top to bottom, shows the original image, reconstruction by Info-VAE, and the histogram of the latent variables $\mathbf{z}_1$. Note that the ranges are different for the four histograms shown here. From left to right, the log-likelihood values of $\mathbf{z}_1$ given the prior distribution are: -530, -638, -1616, -4897, and -343340. The autoencoder is not trained on very dark face images with extreme poses. Given these images, the encoder outputs unusual latent variables and because the decoder has not seen hidden variables outside of the prior distribution, it does a poor job of reconstruction.

These samples contributed to most of the errors of the proposed methods. This is to be expected, because there is no face-image with extreme lighting conditions in OULU-NPU (the source dataset used for training the networks) where all faces are uniformly illuminated.

6.5 Analysis of domain shift

Another way to compare the proposed method to the baseline is to visualize the domain shift present between datasets in the learned feature spaces. For this purpose, *t-distributed stochastic neighbor embedding (t-SNE)* (Maaten and G. Hinton, 2008), a specific multi-dimensional scaling method, is used to visualize the features of samples. Feature-vectors, extracted from samples of each dataset, are projected onto two dimensions using t-SNE and are visualized in figure 6.5. The t-SNE plot of the features of the baseline (DeepPixBiS, top plot) and the proposed method (Autoencoder Error, bottom plot) is shown. Feature-vectors for *bona fide* (triangles) and PA samples (circles) in four different datasets (each identified by a different color) are shown in this plot. Considering only the *bona fide* samples, we note that samples from different datasets form distinct clusters in the top plot. That is, *bona fide* samples of

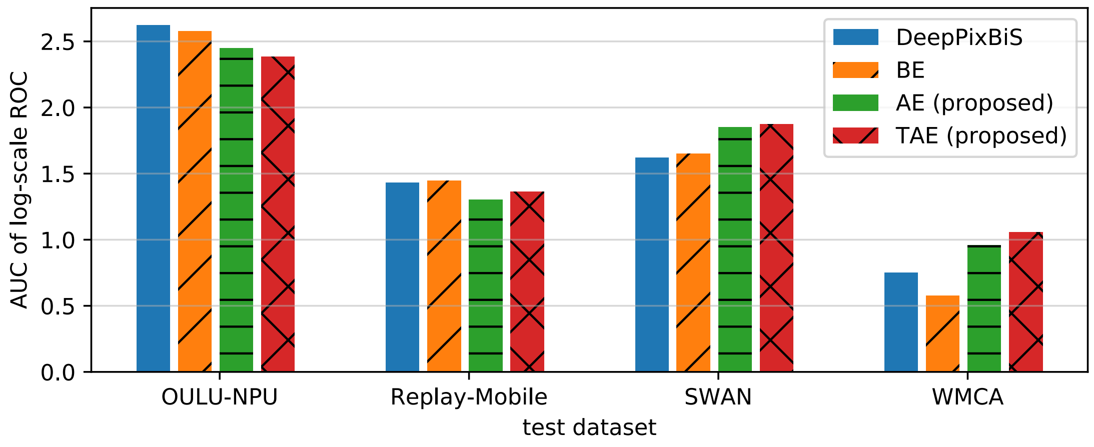
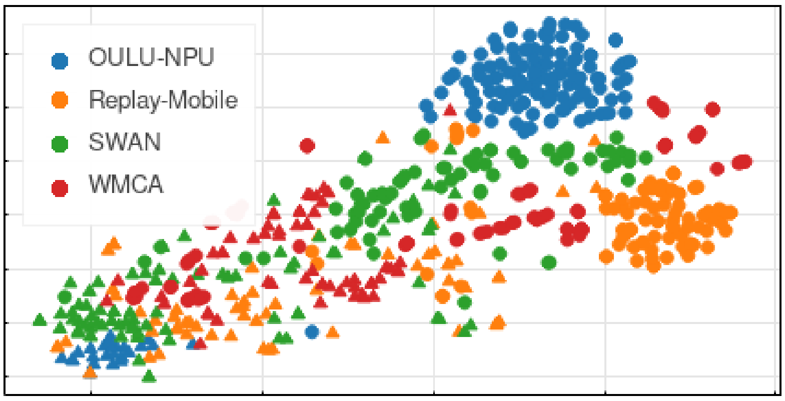


Figure 6.4 – Performance evaluation of the proposed method. The higher the value the better is the performance of the system. The dataset that the model was tested on is shown on the horizontal-axis.

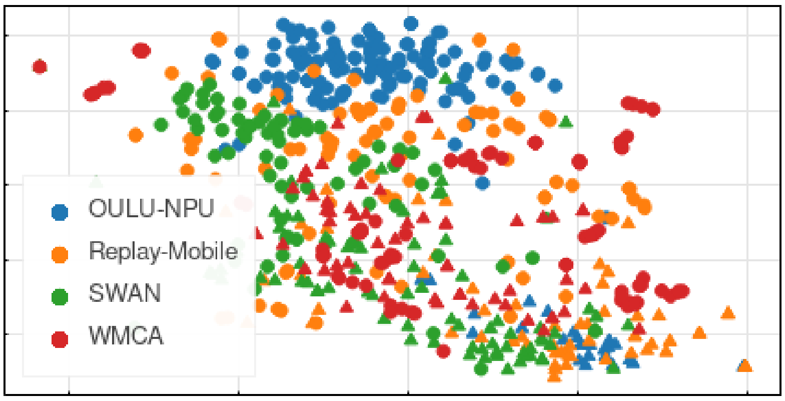
different datasets produce feature-vectors with different distributions. Similar observations can be made for the PA samples of the different datasets as well in the top plot. In the bottom plot, we observe that, unlike the top plot, embeddings corresponding to samples of the *bona fide* class from the different datasets are mixed together. This suggests that the features in the Autoencoder Error method are less sensitive to domain shift compared to the baseline. For the PA class, however, the embeddings still form fairly compact clusters by dataset. One reason for this phenomenon may be as follows. In this work, I have not explicitly tried to suppress the effect of $\mathbf{z}_2$, the latent variable representing the ensemble of nuisance factors exclusive to PAs. Therefore, the factors influencing $\mathbf{z}_2$ may still cause the PA embeddings to form compact clusters.

6.6 Conclusions

I have presented a novel approach to improving generalization of face PAD by taking advantage of large public FR datasets which contain millions of *bona fide* face images from varied sources. I hypothesize that all the factors (variability) present in face images of an FR dataset are nuisance factors to PAD systems. By explicitly modeling these factors using an Info-VAE (an autoencoder which learns meaningful and disentangled representations of data), invariance of these factors is induced to a PAD system. This is done by reconstructing face images with the pre-trained autoencoder and using the reconstruction-error image (the difference between the original image and the reconstructed one) as input to the face PAD system. I assume here that the face image reconstructed by the autoencoder only contains information about the nuisance factors of PAD. When the baseline PAD system is trained on the reconstruction-error images, the intra-dataset performance degrades slightly, but the cross-dataset performance improves significantly for two out of three test datasets. This proves my hypothesis that the



(a) – t-SNE plot of features of DeepPixBiS



(b) – t-SNE plot of features of Autoencoder Error

Figure 6.5 – t-SNE (Maaten and G. Hinton, 2008) plot of the embeddings of two systems: DeepPixBiS (figure 6.5a) and Autoencoder Error (figure 6.5b) using samples from four datasets. Samples with the same color belong to the same face PAD dataset. Triangles are BF samples and circles are PA samples. In figure 6.5a, we observe that, within each class, samples from each dataset are clustered. This is attributed to the *domain shift* present between face PAD datasets. Moreover, we observe that the features of each class and dataset are less clustered in figure 6.5b compared to figure 6.5a.

influence of some nuisance factors on a face PAD system can be lowered by incorporating knowledge from an FR dataset. Furthermore, using the Info-VAE allows us to systematically reject low quality samples, which also contributes to the improved performance.

However, the cross-dataset performance of face PAD methods is still significantly worse compared to the corresponding intra-dataset performance. This implies that the proposed method, does not entirely remove the influence of nuisance factors present in our PAD datasets. To overcome this generalization issue, either larger more varied PAD datasets are needed or we need to restrict our model evaluations to controlled settings. For example, knowing that the proposed method systematically failed on face images with adverse lighting in Replay-Mobile (since all face images were well illuminated in the training dataset, OULU-NPU), we could limit the evaluations to only well illuminated faces.

7 Summary and Future Directions

In this thesis, I have addressed the generalization of face presentation attack detection (PAD).

Convolutional neural network (CNN) based face recognition (FR) systems have shown numerous improvements in terms of performance compared to traditional FR systems that use hand-crafted features. However, they are vulnerable to presentation attacks (PA). In chapter 3, I have showed empirically that state-of-the-art FR systems are highly vulnerable to PAs. Their high vulnerability to PAs calls for development of face PAD methods.

PAs are created by presenting a presentation attack instrument (PAI) (*e.g.,* a printed face photo) to the capture module (a camera sensor) of a biometric system. The process of creating a PAI introduces some artifacts that PAD methods rely on for detecting the PAs. Initial Face PAD systems used hand-crafted features to detect face PA images using several cues. These cues included surface texture analysis of skin, motion analysis (eye blinking, involuntary head movements), and image quality analysis such as color distortion analysis. State-of-the-art face PAD systems are developed using CNNs. CNNs automatically learn appropriate features for the given task (here, PAD).

Many different CNN architectures have been proposed for face PAD. However, no systematic evaluation of CNN architectures has been done before. In chapter 4, I have evaluated the classification performance of six CNN architectures on four face PAD datasets. The evaluation was done in terms of both intra-dataset and cross-dataset classification performance. CNN architectures that classify patches of face images instead of the whole face image were shown to perform consistently better. Moreover, I have proposed a novel CNN architecture in chapter 4 that classifies patches of face images at different scales. The analysis of images at different scales allowed the architecture to generalize better in cross-dataset evaluations.

Still, the performance of face PAD systems drop significantly in cross-dataset evaluations compared to intra-dataset evaluations. The artifacts present in PAIs appear in the captured images in terms of added noise and deformations. Because face PAD methods rely on the presence of this added noise and deformations to detect PAs, they are very sensitive to variations of other factors. These *nuisance* factors include camera devices, capturing conditions, identities, distance of the subject to the camera, resolution of images, facial-makeup, and types of PAIs. The variation of the nuisance factors between datasets causes a distribution shift in the extracted features of datasets. This distribution shift, known as *domain shift*, is the cause of the performance degradation in cross-dataset evaluations. In other words, the learned features by the face PAD CNNs are not invariant to nuisance factors. Given enough variations of nuisance factors in the training dataset, data-driven machine learning models such as CNNs can learn features that are invariant to nuisance factors.

However, collecting PAD datasets while avoiding bias and introducing sufficient variation of nuisance factors is very difficult because of several reasons. First, collecting biometric data is expensive and is a labor intensive task that involves finding willing participants. Second, PAD systems often face unseen PAs when deployed in real-world. That is the PAI may be of different from the ones that the PAD model was trained on. Finally, we may not be aware of all the possible nuisance factors and may easily introduce a bias in our dataset. For example, it may happen that all non PA samples (*bona fide* samples) are captured in an environment with a white background and all PA samples are captured in an environment with a complex background. For all these reasons, domain shift exists between face PAD datasets where each dataset could represent a *domain*. In fact, current face PAD datasets are collected with only 1 to 10 different camera models, 50 to 150 participants, 2 to 5 different PAI types, and lighting conditions with limited variation.

In machine learning, most models are designed under the assumption that the distribution of data does not change between training and evaluation. When a model is trained on a *source* domain and is being evaluated on a *target* domain, two possible situations exist. In the first situation, some information about the target domain is known *a priori*. For example, some training data from the target domain is available. In the other situation, the target domain is unseen. That is no training data from the target domain is available. Instead, we have training data from multiple source domains. In each situation, a different solution is usually proposed.

When limited training data from the target domain is available, *domain adaptation* methods can be used to compensate for domain shift. However, most domain adaptation methods assume that data from all classes is available in the target domain. This assumption does not hold for PAD because collecting PAs is more expensive compared to collecting *bona fide* samples. Moreover, we cannot predict what type of PAIs the system will be presented with in the future. It is much easier to collect only *bona fide* samples in the target domain. Therefore, I have proposed a novel adaptation method in chapter 5, named *domain guided pruning of neural networks*, where it only uses *bona fide* samples from the target domain. The method works based on the hypothesis that some learned filters in CNNs are domain specific and do not generalize to the target dataset. Pruning these filters leads to higher performance in both intra-dataset and cross-dataset evaluations. The proposed domain adaptation method was applied on two CNN architectures and was tested on four face PAD datasets.

When the target domain is unknown, domain generalization methods allow us to train models that are invariant to domain shifts using data from multiple source domains. They do this by allowing models to learn features that are invariant to nuisance factors. Most domain generalization methods require the nuisance factors to be known, categorized, and labeled in the training data. For example, consider the camera model as a nuisance factor in face PAD. For a typical domain generalization method to be applicable, several requirements exist. First, the camera model must be identified as one of the underlying factors of the data. Second, it must be categorized, for example, to mobile camera models and professional camera models. Finally, all samples of the training data must be labeled according to the categories of the nuisance factor. However, identifying all the possible underlying factors of data is impossible. Moreover, categorization of factors is also a subjective process. For example, categorizing the lighting conditions of face images is a subjective process. Therefore, the domain generalization methods that work in a unsupervised manner (without the need for identification of nuisance factors and their labels) are more appealing.

In chapter 6, I have proposed a method to model a subset of nuisance factors of face PAD in an unsupervised manner. Modeling these nuisance factors allows us to induce invariance to these factors in our PAD models. However, as discussed before, current face PAD datasets are limited in terms of variation of nuisance factors. Therefore, current face PAD datasets cannot be used to adequately model the variations of nuisance factors. Instead, I have proposed to use FR datasets (which contain only *bona fide* samples) to model a subset of the nuisance factors of face PAD. I have hypothesized that all the underlying factors of *bona fide* samples, are also present in PA samples as well. For example, identities, lighting conditions, and many other factors are common nuisance factors present in both *bona fide* and PA samples. Recent FR datasets which are collected from the Internet and contain millions of different face images are a good candidate to model these common nuisance factors. I have used a *information maximizing variational autoencoder (Info-VAE)* to model the underlying factors of *bona fide* face images in an unsupervised manner. Info-VAEs are a class of autoencoders that learn decorrelated factors of data as their latent variable. Because the Info-VAE is trained only on *bona fide* samples, it can only reconstruct face images using only the factors that it has observed during training. That is, when a PA face image is reconstructed using this autoencoder, the reconstructed image will not contain information about the original image being an attack or not. Instead, the error between the original image and the reconstructed image contains this information. Therefore, I have proposed the reconstruction-error image (difference between the original image and the reconstructed image) to be used as input to face PAD models instead of the original image. The proposed method can be seen as a pre-processing method in face PAD systems. The proposed method improved the cross-dataset performance of our face PAD baseline in two out of three tested datasets.

All in all, improving the classification performance of face PAD systems in cross-dataset scenarios remains a challenge that needs further research. Cross-dataset scenarios represent real-world applications of face PAD where the trained PAD model will be tested in a different setting. The proposed domain guided pruning method in chapter 5 and the proposed autoencoder-based method in chapter 6 are a step towards the improvement of the cross-dataset performance of face PAD methods. In this thesis, I have investigated that how the large and varied FR datasets can be used to improve the generalization of face PAD. In fact, because the proposed domain adaptation method in chapter 5 only needs *bona fide* samples from the target domain to work, I have also investigated, in chapter 5, use of FR datasets in this method as well. In chapter 5, I have showed how the proposed domain adaptation method can be seen as a domain generalization method by using *bona fide* samples of an FR dataset for pruning. The proposed method when implemented as domain generalization method either did not degrade or improved the cross-dataset performance of PAD models.

Below, the future directions, publications, and software related to this thesis are listed.

7.1 Future Directions

The following future research directions may be considered given the work done in this thesis:

1. In chapter 3, we observed that current state-of-the-art FR systems are highly vulnerable to presentation attacks. Can we build FR neural networks that not only have high classification performance in FR but also are not vulnerable to presentation attacks? Multi-task learning approaches of FR and PAD are a solution here. These approaches have been investigated by Xiaoguang Tu, J. Zhao, et al. (2019) and Ying, X. Li, and Chuah (2018) but there is still room for improvement.

2. In chapter 4, we evaluated several CNN architectures on four face PAD datasets in terms of both intra-dataset and cross-dataset performance. We observed that patch-based CNN architectures have a higher performance overall. However, the development of neural network architectures is a fast developing field and investigation of novel architectures is needed. For example, testing capsule networks (Sabour, Frosst, and G. E. Hinton, 2017) on face PAD is a good direction.

3. In chapters 5 and 6, we observed the effect of domain shift on the cross-dataset performance of face PAD systems and saw how the proposed domain adaptation and generalization methods account for domain shift. However, it is worth investigating the impact of *data augmentation* methods on reducing the effect of domain shift. There are many data augmentation methods available ranging from traditional fixed methods to learned methods. Several adversarial learning based methods such as (Miyato, Maeda, et al., 2018) have been proposed which can be seen as a kind of data augmentation method. Investigation of their impact on face PAD is well worth the effort in my opinion.

7.2 Related Publications

During the four years of my PhD studies, I have made several contributions in my research field in terms of publications which are listed below.

7.2.1 Book Chapters

- Ivana Chingovska et al. (2019). "Evaluation Methodologies for Biometric Presentation Attack Detection". In: *Handbook of Biometric Anti-Spoofing*. Ed. by Sébastien Marcel et al. 2nd. Springer International Publishing, I am the 2nd author.

- Sushil Bhattacharjee et al. (Apr. 2019). "Recent Advances in Face Presentation Attack Detection". In: *Handbook of Biometric Anti-Spoofing*. Ed. by Sébastien Marcel et al. 2nd. Advances in Computer Vision and Pattern Recognition. Springer, I am the 2nd author.

7.2.2 Journal Papers

- Amir Mohammadi, Sushil Bhattacharjee, and Sébastien Marcel (2017). "Deeply Vulnerable: A Study of the Robustness of Face Recognition to Presentation Attacks". In: *IET Biometrics* 7.1, pp. 15–26

7.2.3 Conference Papers

- Amir Mohammadi, Sushil Bhattacharjee, and Sébastien Marcel (Jan. 2020b). "Improving Cross-Dataset Performance Of Face Presentation Attack Detection Systems Using Face Recognition Datasets". In: *45th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2020)*. IEEE

- Amir Mohammadi, Sushil Bhattacharjee, and Sébastien Marcel (Jan. 2020a). "Domain Adaptation For Generalization Of Face Presentation Attack Detection In Mobile Settings With Minimal Information". In: *45th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2020)*. IEEE

- Olegs Nikisins et al. (2018). "On Effectiveness of Anomaly Detection Approaches against Unseen Presentation Attacks in Face Anti-Spoofing". In: *The 11th IAPR International Conference on Biometrics (ICB 2018)*, I am the 2nd author.

- Sushil Bhattacharjee, Amir Mohammadi, and Sébastien Marcel (Oct. 2018). "Spoofing Deep Face Recognition With Custom Silicone Masks". In: *Proceedings of BTAS2018*

- Milos Cernak et al. (Aug. 2017). "Bob Speaks Kaldi". In: *Proc. of Interspeech*, I am the 3rd author.

- Z. Boulkenafet et al. (Oct. 2017). "A Competition on Generalized Software-Based Face

Presentation Attack Detection in Mobile Scenarios". In: *Proceedings of the International Joint Conference on Biometrics, 2017*, I am the 8th author.

- André Anjos et al. (Aug. 2017). "Continuously Reproducing Toolchains in Pattern Recognition and Machine Learning Experiments". In: *Thirty-Fourth International Conference on Machine Learning*, I am the 5th author.

7.2.4 Pre-print Publications

- Raghavendra Ramachandra et al. (Dec. 2019). "Smartphone Multi-Modal Biometric Authentication: Database and Evaluation". In: *arXiv:1912.02487 [cs]*, I am the 3rd author.

7.3 Related Software

Software was an integral part of this work. Had it not been for the substantial amount of machine learning and computational software (*e.g.,* Bob (Anjos et al., 2012; Anjos et al., 2017), Tensorflow (Abadi et al., 2016; Abadi et al., 2016), and Conda (Analytics, 2019)) readily available during my work, this thesis would not have been possible. I have mainly developed my work on top of the signal processing and machine learning toolkit *Bob* (Anjos et al., 2012; Anjos et al., 2017)[1]. My work has been done following the reproducible research philosophy by Anjos et al. (2017) who emphasize that a reproducible research work should be repeatable, shareable, extensible, and stable. Bob is made of over 100 packages where each package is related to a specific task. Below, is the list of Bob packages that I have contributed to and is related to this thesis.

- `bob.thesis.amohammadi`: created by me, allows to reproduce all the experiments done in this thesis.
- `bob.pad.face`: originally developed by me, this Bob package implements a complete ecosystem for developing face PAD methods. It contains the implementation of many face PAD baselines and ready made interfaces to face PAD datasets.
- `bob.db.swan`: Bob database interface for the SWAN dataset (Ramachandra et al., 2019).
- `bob.db.oulunpu`: Bob database interface for the OULU-NPU dataset (Zinelabinde Boulkenafet et al., 2017).
- `bob.db.replaymobile`: Bob database interface for the Replay-Mobile dataset (Costa-Pazo et al., 2016).
- `bob.db.batl`: Bob database interface for the WMCA dataset (George et al., 2019).

[1] https://www.idiap.ch/software/bob/

- `bob.db.replay`: Bob database interface for the Replay-Attack dataset (Chingovska, Anjos, and Marcel, 2012).
- `bob.db.msu_mfsd_mod`: Bob database interface for the MSU-MFSD dataset (Wen, H. Han, and Jain, 2015).
- `bob.db.mobio`: Bob database interface for the MOBIO dataset (McCool et al., 2012).

A Details of the CNN Architectures

Details of the CNN architecture blocks introduced in section 2.2 are given here.

Table A.1 – Architecture details of inception (3a) module presented in (Szegedy et al., 2015). See figure 2.4 on page 17 for a better representation of this block. The input to the architecture are feature maps of the preceding layer of size 28 × 28 × 192.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
branch1_conv1 (Conv2D)	filters=64, kernel_size=(1, 1), strides=(1, 1), padding=same, activation=relu	(28, 28, 64)	12,352	input
branch2_conv1 (Conv2D)	filters=96, kernel_size=(1, 1), strides=(1, 1), padding=same, activation=relu	(28, 28, 96)	18,528	input
branch2_conv2 (Conv2D)	filters=128, kernel_size=(3, 3), strides=(1, 1), padding=same, activation=relu	(28, 28, 128)	110,720	branch2_conv1
branch3_conv1 (Conv2D)	filters=16, kernel_size=(1, 1), strides=(1, 1), padding=same, activation=relu	(28, 28, 16)	3,088	input
branch3_conv2 (Conv2D)	filters=32, kernel_size=(5, 5), strides=(1, 1), padding=same, activation=relu	(28, 28, 32)	12,832	branch3_conv1
branch4_pool1 (MaxPooling2D)	pool_size=(3, 3), strides=(1, 1), padding=same	(28, 28, 192)	0	input
branch4_conv1 (Conv2D)	filters=32, kernel_size=(1, 1), strides=(1, 1), padding=same, activation=relu	(28, 28, 32)	6,176	branch4_pool1
concat (Concatenate)	axis=-1	(28, 28, 256)	0	branch1_conv1 branch2_conv2 branch3_conv2 branch4_conv1
Model	Parameters: total=163,696, trainable=163,696			

Table A.2 – Details of the *conv2d_bn* (Conv2D_BN) block in the InceptionResNetV2 architecture shown in table 2.7 on page 23. The parameters of a Conv2D_BN block in table 2.7 correspond to the parameters of the Conv2D layer in this table.

Layer (type)	Details	Output Shape	Number of Parameters
conv2d (Conv2D)	filters=32, kernel_size=(3, 3), strides=(2, 2), padding=valid	(149, 149, 32)	864
batch_normalization (BatchNormalization)		(149, 149, 32)	96
activation (Activation)	activation=relu	(149, 149, 32)	0
Model	Parameters: total=960, trainable=896		

Table A.3 – Details of the *inception_a* (InceptionA) block in the InceptionResNetV2 architecture shown in table 2.7 on page 23. The *pool_filters* parameter of an InceptionA block in table 2.7 corresponds to the number of filters in the branch4_conv1 layer.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
branch1_conv1 (Conv2D_BN)	filters=96, kernel_size=1, strides=1	(35, 35, 96)	18,720	input
branch2_conv1 (Conv2D_BN)	filters=64, kernel_size=1, strides=1	(35, 35, 64)	12,480	input
branch2_conv2 (Conv2D_BN)	filters=96, kernel_size=3, strides=1	(35, 35, 96)	55,584	branch2_conv1
branch2_conv3 (Conv2D_BN)	filters=96, kernel_size=3, strides=1	(35, 35, 96)	83,232	branch2_conv2
branch3_conv1 (Conv2D_BN)	filters=48, kernel_size=1, strides=1	(35, 35, 48)	9,360	input
branch3_conv2 (Conv2D_BN)	filters=64, kernel_size=5, strides=1	(35, 35, 64)	76,992	branch3_conv1
branch4_pool1 (AveragePooling2D)	pool_size=(3, 3), strides=(1, 1), padding=same	(35, 35, 192)	0	input
branch4_conv1 (Conv2D_BN)	filters=64, kernel_size=1, strides=1	(35, 35, 64)	12,480	branch4_pool1
concatenate (Concatenate)	axis=3	(35, 35, 320)	0	branch1_conv1
				branch2_conv3
				branch3_conv2
				branch4_conv1
Model	Parameters: total=268,848, trainable=267,792			

Table A.4 – Details of the *block35_1* (InceptionResnetBlock) block in the InceptionResNetV2 architecture shown in table 2.7 on page 23. The *scale* parameter of an InceptionResnetBlock block in table 2.7 corresponds to the *scale* value in the *scaled_residual* layer and the α value in equation (2.15). The number of filters in convolutional layers are constant in an InceptionResnetBlock except that all numbers may be divided by an integer n. The n parameter of the InceptionResnet-Blocks are given in table 2.7.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
branch0_conv1 (Conv2D_BN)	filters=32, kernel_size=1, strides=1	(35, 35, 32)	10,336	input
branch1_conv1 (Conv2D_BN)	filters=32, kernel_size=1, strides=1	(35, 35, 32)	10,336	input
branch1_conv2 (Conv2D_BN)	filters=32, kernel_size=3, strides=1	(35, 35, 32)	9,312	branch1_conv1
branch2_conv1 (Conv2D_BN)	filters=32, kernel_size=1, strides=1	(35, 35, 32)	10,336	input
branch2_conv2 (Conv2D_BN)	filters=48, kernel_size=3, strides=1	(35, 35, 48)	13,968	branch2_conv1
branch2_conv3 (Conv2D_BN)	filters=64, kernel_size=3, strides=1	(35, 35, 64)	27,840	branch2_conv2
concatenate (Concatenate)	axis=3	(35, 35, 128)	0	branch0_conv1
				branch1_conv2
				branch2_conv3
up_conv (Conv2D_BN)	filters=320, kernel_size=1, strides=1	(35, 35, 320)	41,280	concatenate
scaled_residual (ScaledResidual)	scale=0.17	(35, 35, 320)	0	input
				up_conv
act (Activation)	activation=relu	(35, 35, 320)	0	scaled_residual
Model	Parameters: total=123,408, trainable=122,928			

Table A.5 – Details of the *block17_1* (InceptionResnetBlock) block in the InceptionResNetV2 architecture shown in table 2.7 on page 23. The parameters of an InceptionResnetBlock is explained in table A.4.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
branch0_conv1 (Conv2D_BN)	filters=192, kernel_size=1, strides=1	(17, 17, 192)	209,472	input
branch1_conv1 (Conv2D_BN)	filters=128, kernel_size=1, strides=1	(17, 17, 128)	139,648	input
branch1_conv2 (Conv2D_BN)	filters=160, kernel_size=(1, 7), strides=1	(17, 17, 160)	143,840	branch1_conv1
branch1_conv3 (Conv2D_BN)	filters=192, kernel_size=(7, 1), strides=1	(17, 17, 192)	215,616	branch1_conv2
concatenate (Concatenate)	axis=3	(17, 17, 384)	0	branch0_conv1
				branch1_conv3
up_conv (Conv2D_BN)	filters=1088, kernel_size=1, strides=1	(17, 17, 1088)	418,880	concatenate
scaled_residual (ScaledResidual)	scale=0.1	(17, 17, 1088)	0	input
				up_conv
act (Activation)	activation=relu	(17, 17, 1088)	0	scaled_residual
Model	Parameters: total=1,127,456, trainable=1,126,112			

Table A.6 – Details of the *block8_1* (InceptionResnetBlock) block in the InceptionResNetV2 architecture shown in table 2.7 on page 23. The parameters of an InceptionResnetBlock is explained in table A.4.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
branch0_conv1 (Conv2D_BN)	filters=192, kernel_size=1, strides=1	(8, 8, 192)	399,936	input
branch1_conv1 (Conv2D_BN)	filters=192, kernel_size=1, strides=1	(8, 8, 192)	399,936	input
branch1_conv2 (Conv2D_BN)	filters=224, kernel_size=(1, 3), strides=1	(8, 8, 224)	129,696	branch1_conv1
branch1_conv3 (Conv2D_BN)	filters=256, kernel_size=(3, 1), strides=1	(8, 8, 256)	172,800	branch1_conv2
concatenate (Concatenate)	axis=3	(8, 8, 448)	0	branch0_conv1 branch1_conv3
up_conv (Conv2D_BN)	filters=2080, kernel_size=1, strides=1	(8, 8, 2080)	933,920	concatenate
scaled_residual (ScaledResidual)	scale=0.2	(8, 8, 2080)	0	input up_conv
act (Activation)	activation=relu	(8, 8, 2080)	0	scaled_residual
Model	Parameters: total=2,036,288, trainable=2,034,560			

Table A.7 – Details of the *reduction_a* (ReductionA) block in the InceptionResNetV2 architecture shown in table 2.7 on page 23. The parameters of a ReductionA block as shown in table 2.7 have the following meaning. n corresponds to the number of filters of the convolutional layer in branch 1. k, kl, and km correspond to the number of filters of the convolutional layers in branch 2.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
branch1_conv1 (Conv2D_BN)	filters=384, kernel_size=3, strides=2	(17, 17, 384)	1,107,072	input
branch2_conv1 (Conv2D_BN)	filters=256, kernel_size=1, strides=1	(35, 35, 256)	82,688	input
branch2_conv2 (Conv2D_BN)	filters=256, kernel_size=3, strides=1	(35, 35, 256)	590,592	branch2_conv1
branch2_conv3 (Conv2D_BN)	filters=384, kernel_size=3, strides=2	(17, 17, 384)	885,888	branch2_conv2
branch3_pool1 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(17, 17, 320)	0	input
reduction_a/mixed (Concatenate)	axis=3	(17, 17, 1088)	0	branch1_conv1 branch2_conv3 branch3_pool1
Model	Parameters: total=2,666,240, trainable=2,663,680			

Table A.8 – Details of the *reduction_b* (ReductionB) block in the InceptionResNetV2 architecture shown in table 2.7 on page 23. The parameters of a ReductionB block as shown in table 2.7 have the following meaning. n and no correspond to the number of filters of the convolutional layers in branch 1. p and pq correspond to the number of filters of the convolutional layers in branch 2. k, kl, and km correspond to the number of filters of the convolutional layers in branch 3.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
branch1_conv1 (Conv2D_BN)	filters=256, kernel_size=1, strides=1	(17, 17, 256)	279,296	input
branch1_conv2 (Conv2D_BN)	filters=384, kernel_size=3, strides=2	(8, 8, 384)	885,888	branch1_conv1
branch2_conv1 (Conv2D_BN)	filters=256, kernel_size=1, strides=1	(17, 17, 256)	279,296	input
branch2_conv2 (Conv2D_BN)	filters=288, kernel_size=3, strides=2	(8, 8, 288)	664,416	branch2_conv1
branch3_conv1 (Conv2D_BN)	filters=256, kernel_size=1, strides=1	(17, 17, 256)	279,296	input
branch3_conv2 (Conv2D_BN)	filters=288, kernel_size=3, strides=1	(17, 17, 288)	664,416	branch3_conv1
branch3_conv3 (Conv2D_BN)	filters=320, kernel_size=3, strides=2	(8, 8, 320)	830,400	branch3_conv2
branch4_pool1 (MaxPooling2D)	pool_size=(3, 3), strides=(2, 2), padding=valid	(8, 8, 1088)	0	input
reduction_b/mixed (Concatenate)	axis=3	(8, 8, 2080)	0	branch1_conv2 branch2_conv2 branch3_conv3 branch4_pool1
Model	Parameters: total=3,883,008, trainable=3,878,912			

Table A.9 – Details of *dense_block_1* of DenseNet-161 (G. Huang et al., 2017) as shown in table 2.8 on page 24. Also, see figure 2.6 on page 21 for a schematic of a DenseBlock. The input is 56 × 56 × 96 feature maps from the preceding layer. Details of *conv_block_1* is shown in table A.10.

Layer (type)	Details	Output Shape	Number of Parameters	Connected to
conv_block_1 (ConvBlock)	filters=48, bottleneck=True, dropout_rate=0	(56, 56, 48)	102,528	input
concat_1 (Concatenate)	axis=-1	(56, 56, 144)	0	input
				conv_block_1
conv_block_2 (ConvBlock)	filters=48, bottleneck=True, dropout_rate=0	(56, 56, 48)	111,936	concat_1
concat_2 (Concatenate)	axis=-1	(56, 56, 192)	0	concat_1
				conv_block_2
conv_block_3 (ConvBlock)	filters=48, bottleneck=True, dropout_rate=0	(56, 56, 48)	121,344	concat_2
concat_3 (Concatenate)	axis=-1	(56, 56, 240)	0	concat_2
				conv_block_3
conv_block_4 (ConvBlock)	filters=48, bottleneck=True, dropout_rate=0	(56, 56, 48)	130,752	concat_3
concat_4 (Concatenate)	axis=-1	(56, 56, 288)	0	concat_3
				conv_block_4
conv_block_5 (ConvBlock)	filters=48, bottleneck=True, dropout_rate=0	(56, 56, 48)	140,160	concat_4
concat_5 (Concatenate)	axis=-1	(56, 56, 336)	0	concat_4
				conv_block_5
conv_block_6 (ConvBlock)	filters=48, bottleneck=True, dropout_rate=0	(56, 56, 48)	149,568	concat_5
concat_6 (Concatenate)	axis=-1	(56, 56, 384)	0	concat_5
				conv_block_6
Model	Parameters: total=756,288, trainable=751,392			

Table A.10 – Details of the first ConvBlock, *conv_block_1*, in the first DenseBlock (see table A.9) of DenseNet-161 (G. Huang et al., 2017) which is shown in table 2.8 on page 24. If *bottleneck* (a parameter that changes ConvBlocks) is *True*, *conv1*, *norm2*, and *relu2* are added. *conv1* always has four times the number of filters of *conv2*. The number of filters in *conv2* is the *growth rate*. The input is 56 × 56 × 96 feature maps from the preceding layer.

Layer (type)	Details	Output Shape	Number of Parameters
norm1 (BatchNormalization)		(56, 56, 96)	384
relu1 (Activation)	activation=relu	(56, 56, 96)	0
conv1 (Conv2D)	filters=192, kernel_size=(1, 1), strides=(1, 1), padding=valid	(56, 56, 192)	18,432
norm2 (BatchNormalization)		(56, 56, 192)	768
relu2 (Activation)	activation=relu	(56, 56, 192)	0
conv2_pad (ZeroPadding2D)	padding=((1, 1), (1, 1))	(58, 58, 192)	0
conv2 (Conv2D)	filters=48, kernel_size=(3, 3), strides=(1, 1), padding=valid	(56, 56, 48)	82,944
dropout (Dropout)	drop_rate=0	(56, 56, 48)	0
Model	Parameters: total=102,528, trainable=101,952		

Table A.11 – Details of *transition_block_1* of DenseNet-161 (G. Huang et al., 2017) as shown in table 2.8 on page 24. The input is 56 × 56 × 384 feature maps from the preceding layer.

Layer (type)	Details	Output Shape	Number of Parameters
norm (BatchNormalization)		(56, 56, 384)	1,536
relu (Activation)	activation=relu	(56, 56, 384)	0
conv (Conv2D)	filters=192, kernel_size=(1, 1), strides=(1, 1), padding=valid	(56, 56, 192)	73,728
pool (AveragePooling2D)	pool_size=(2, 2), strides=(2, 2), padding=valid	(28, 28, 192)	0
Model	Parameters: total=75,264, trainable=74,496		

Bibliography

Abadi, Martín et al. (2016). “Tensorflow: A System for Large-Scale Machine Learning”. In: *12th Symposium on Operating Systems Design and Implementation (16)*, pp. 265–283 (cit. on p. 126).

Almeida, Waldir Rodrigues de (2018). “Data-Driven Face Presentation-Attack Detection in Mobile Devices”. PhD thesis (cit. on pp. 35, 81).

Amos, B., B. Ludwiczuk, and M. Satyanarayanan (2016). *OpenFace: A General-Purpose Face Recognition Library with Mobile Applications*. Tech. rep. CMU-CS-16-118. CMU School of Computer Science (cit. on p. 64).

Analytics, Continuum (2019). *Conda: A Cross-Platform, Python-Agnostic Binary Package Manager* (cit. on p. 126).

Anjos, André, Manuel Günther, Tiago de Freitas Pereira, Pavel Korshunov, Amir Mohammadi, and Sébastien Marcel (Aug. 2017). “Continuously Reproducing Toolchains in Pattern Recognition and Machine Learning Experiments”. In: *Thirty-Fourth International Conference on Machine Learning* (cit. on pp. 68, 126).

Anjos, André and Sébastien Marcel (2011). “Counter-Measures to Photo Attacks in Face Recognition: A Public Database and a Baseline”. In: *Biometrics (IJCB), 2011 International Joint Conference On*. IEEE, pp. 1–7 (cit. on pp. 4, 35).

Anjos, André, Laurent El-Shafey, Roy Wallace, Manuel Günther, Christopher McCool, and Sébastien Marcel (2012). “Bob: A Free Signal Processing and Machine Learning Toolbox for Researchers”. In: *Proceedings of the 20th ACM International Conference on Multimedia*. ACM, pp. 1449–1452 (cit. on pp. 68, 126).

Atoum, Yousef, Yaojie Liu, Amin Jourabloo, and Xiaoming Liu (Oct. 2017). “Face Anti-Spoofing Using Patch and Depth-Based CNNs”. In: *2017 IEEE International Joint Conference on Biometrics (IJCB)*. Denver, CO: IEEE, pp. 319–328 (cit. on pp. 5, 35, 37, 38, 81–83, 86).

Bengio, Samy, Johnny Mariéthoz, and Mikaela Keller (2005). “The Expected Performance Curve”. In: *International Conference on Machine Learning, ICML, Workshop on ROC Analysis in Machine Learning* (cit. on pp. 53, 70).

Bengio, Yoshua (2009). “Learning Deep Architectures for AI”. In: *Foundations and trends® in Machine Learning* 2.1. 03079, pp. 1–127 (cit. on p. 25).

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). “Representation Learning: A Review and New Perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8, pp. 1798–1828 (cit. on pp. 6, 25, 110).

Bhattacharjee, Sushil, Amir Mohammadi, André Anjos, and Sébastien Marcel (Apr. 2019). "Recent Advances in Face Presentation Attack Detection". In: *Handbook of Biometric Anti-Spoofing*. Ed. by Sébastien Marcel, Mark Nixon, Julian Fierrez, and Nicholas Evans. 2nd. Advances in Computer Vision and Pattern Recognition. Springer (cit. on pp. 35, 125).

Bhattacharjee, Sushil, Amir Mohammadi, and Sébastien Marcel (Oct. 2018). "Spoofing Deep Face Recognition With Custom Silicone Masks". In: *Proceedings of BTAS2018* (cit. on p. 125).

Boulkenafet, Z. et al. (Oct. 2017). "A Competition on Generalized Software-Based Face Presentation Attack Detection in Mobile Scenarios". In: *Proceedings of the International Joint Conference on Biometrics, 2017* (cit. on pp. 5, 35, 81, 90, 125).

Boulkenafet, Zinelabinde, Jukka Komulainen, Lei Li, Xiaoyi Feng, and Abdenour Hadid (2017). "OULU-NPU: A Mobile Face Presentation Attack Database with Real-World Variations". In: *Automatic Face & Gesture Recognition (FG 2017), 2017 12th IEEE International Conference On*. IEEE, pp. 612–618 (cit. on pp. 59–61, 114, 126).

Cardinaux, Fabien, Conrad Sanderson, and Samy Bengio (2006). "User Authentication via Adapted Statistical Models of Face Images". In: *IEEE Transactions on Signal Processing* 54.1. 00118, pp. 361–373 (cit. on p. 112).

Cernak, Milos, Alain Komaty, Amir Mohammadi, André Anjos, and Sébastien Marcel (Aug. 2017). "Bob Speaks Kaldi". In: *Proc. of Interspeech* (cit. on p. 125).

Chingovska, Ivana, André Anjos, and Sébastien Marcel (2012). "On the Effectiveness of Local Binary Patterns in Face Anti-Spoofing". In: *Biometrics Special Interest Group (BIOSIG), 2012 BIOSIG-Proceedings of the International Conference of The*. IEEE, pp. 1–7 (cit. on pp. 4, 35, 46, 47, 59, 65, 67, 127).

Chingovska, Ivana, André Anjos, and Sébastien Marcel (2014). "Biometrics Evaluation under Spoofing Attacks". In: *Information Forensics and Security, IEEE Transactions on* 9.12, pp. 2264–2276 (cit. on pp. 50, 56, 57).

Chingovska, Ivana, Amir Mohammadi, André Anjos, and Sébastien Marcel (2019). "Evaluation Methodologies for Biometric Presentation Attack Detection". In: *Handbook of Biometric Anti-Spoofing*. Ed. by Sébastien Marcel, Mark Nixon, Julian Fierrez, and Nicholas Evans. 2nd. Springer International Publishing (cit. on pp. 48–50, 56, 57, 125).

Costa-Pazo, Artur, Sushil Bhattacharjee, Esteban Vazquez-Fernandez, and Sébastien Marcel (2016). "The REPLAY-MOBILE Face Presentation-Attack Database". In: *Biometrics Special Interest Group (BIOSIG), 2016 International Conference of The*. IEEE, pp. 1–7 (cit. on pp. 59, 60, 65, 114, 126).

Csurka, Gabriela (2017). "Domain Adaptation for Visual Applications: A Comprehensive Survey". In: *arXiv preprint arXiv:1702.05374* (cit. on p. 109).

Duc, Nguyen Minh and Bui Quang Minh (2009). "Your Face Is Not Your Password Face Authentication Bypassing Lenovo–Asus–Toshiba". In: *Black Hat Briefings* (cit. on pp. 3, 64).

El Shafey, Laurent, Chris McCool, Roy Wallace, and Sébastien Marcel (2013). "A Scalable Formulation of Probabilistic Linear Discriminant Analysis: Applied to Face Recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 35.7, pp. 1788–1794 (cit. on p. 27).

Erdogmus, Nesli and Jean-Luc Dugelay (May 2012). “On Discriminative Properties of TPS Warping Parameters for 3D Face Recognition”. In: *Proc. Intl. Conf. on Informatics, Electronics and Vision (ICIEV)* (cit. on p. 64).

Galbally, Javier, Sébastien Marcel, and Julian Fierrez (2014). “Image Quality Assessment for Fake Biometric Detection: Application to Iris, Fingerprint, and Face Recognition”. In: *IEEE transactions on image processing* 23.2, pp. 710–724 (cit. on pp. 4, 35).

Ganin, Yaroslav and Victor Lempitsky (2014). “Unsupervised Domain Adaptation by Backpropagation”. In: *arXiv preprint arXiv:1409.7495* (cit. on p. 44).

Ganin, Yaroslav et al. (2016). “Domain-Adversarial Training of Neural Networks”. In: *The Journal of Machine Learning Research* 17.1, pp. 2096–2030 (cit. on pp. 6, 110, 112).

George, Anjith and Sébastien Marcel (2019). “Deep Pixel-Wise Binary Supervision for Face Presentation Attack Detection”. In: *International Conference on Biometrics* (cit. on pp. 5, 35, 38, 39, 81, 82, 94, 110, 115).

George, Anjith, Zohreh Mostaani, David Geissenbuhler, Olegs Nikisins, André Anjos, and Sébastien Marcel (2019). “Biometric Face Presentation Attack Detection with Multi-Channel Convolutional Neural Network”. In: *IEEE Transactions on Information Forensics and Security* (cit. on pp. 4, 59, 62, 114, 126).

Glembek, O., L. Burget, N. Dehak, N. Brummer, and P. Kenny (2009). “Comparison of Scoring Methods Used in Speaker Recognition with Joint Factor Analysis”. In: *Proc. IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4057–4060 (cit. on p. 33).

Glorot, Xavier and Yoshua Bengio (2010). “Understanding the Difficulty of Training Deep Feedforward Neural Networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256 (cit. on p. 18).

Gomez-Barrero, Marta, Javier Galbally, Julian Fierrez, and Javier Ortega-Garcia (2013). “Multimodal Biometric Fusion: A Study on Vulnerabilities to Indirect Attacks”. In: *Iberoamerican Congress on Pattern Recognition.* Springer, pp. 358–365 (cit. on p. 34).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning.* Book in preparation for MIT Press (cit. on pp. xiii, 9, 12, 14, 29, 89).

Goodfellow, Ian et al. (2014). “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*, pp. 2672–2680 (cit. on pp. 25, 45, 46).

Gretton, Arthur, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola (2012). “A Kernel Two-Sample Test”. In: *Journal of Machine Learning Research* 13.Mar, pp. 723–773 (cit. on p. 44).

Grother, Patrick, Mei Ngan, and Kayee Hanaoka (Apr. 2017). *Face Recognition Vendor Test (FRVT) Part 1: Verification.* Tech. rep. NIST (cit. on pp. 33, 70).

Günther, Manuel, Laurent El Shafey, and Sébastien Marcel (Feb. 2016). “Face Recognition in Challenging Environments: An Experimental and Reproducible Research Survey”. In: *Face Recognition across the Imaging Spectrum.* Ed. by Thirimachos Bourlai. 1st ed. Springer (cit. on p. 69).

Guo, Yandong, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao (2016). “MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition”. In: *arXiv preprint arXiv:1607.08221* (cit. on pp. 6, 31, 110, 115).

Hadid, Abdenour (2014). “Face Biometrics Under Spoofing Attacks: Vulnerabilities, Countermeasures, Open Issues, and Research Directions”. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops*. IEEE, pp. 113–118. DOI: 10.1109/CVPRW.2014.22 (cit. on pp. 3, 64).

Hao, Huiling and Mingtao Pei (2019). “Face Liveness Detection Based on Client Identity Using Siamese Network”. In: *arXiv preprint arXiv:1903.05369* (cit. on p. 81).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778 (cit. on pp. 19–21, 81).

Hernandez-Ortega, Javier, Julian Fierrez, Aythami Morales, and Javier Galbally (2019). “Introduction to Face Presentation Attack Detection”. en. In: *Handbook of Biometric Anti-Spoofing: Presentation Attack Detection*. Ed. by Sébastien Marcel, Mark S. Nixon, Julian Fierrez, and Nicholas Evans. Advances in Computer Vision and Pattern Recognition. Cham: Springer International Publishing, pp. 187–206. DOI: 10.1007/978-3-319-92627-8_9 (cit. on p. 34).

Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov (2012). “Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors”. In: *arXiv preprint arXiv:1207.0580* (cit. on pp. 14, 15).

Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger (2017). “Densely Connected Convolutional Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708 (cit. on pp. 21, 22, 24, 84, 115, 132).

Huang, Gary B., Manu Ramesh, Tamara Berg, and Erik Learned-Miller (2007). *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. Technical Report 07-49, University of Massachusetts, Amherst (cit. on pp. 3, 27, 30, 33).

Ioffe, Sergey and Christian Szegedy (Feb. 2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *arXiv:1502.03167 [cs]* (cit. on pp. 17, 19–21).

ISO/IEC 19795-1 (Apr. 2006). *ISO/IEC 19795-1:2006 Information Technology - Biometric Performance Testing and Reporting - Part 1: Principles and Framework*. en. ISO ISO/IEC 19795-1:2006. International Organization for Standardization, p. 56 (cit. on p. 55).

ISO/IEC DIS 30107-1 (Jan. 2016). *ISO/IEC DIS 30107-1. Information Technology – Biometric Presentation Attack Detection – Part 1: Framework*. Standard. Geneva, CH: International Organization for Standardization (cit. on pp. 3, 27, 34, 48).

ISO/IEC DIS 30107-3 (Jan. 2016). *ISO/IEC DIS 30107-3. Information Technology – Biometric Presentation Attack Detection – Part 3: Testing and Reporting*. Standard. Geneva, CH: International Organization for Standardization (cit. on pp. 55, 56).

Jain, Anil, Patrick Flynn, and Arun Ross (2007). *Handbook of Biometrics*. Springer Science & Business Media (cit. on p. 1).

Jain, Anil, Arun Ross, and Salil Prabhakar (2004). “An Introduction to Biometric Recognition”. In: *IEEE Transactions on circuits and systems for video technology* 14.1 (cit. on pp. 1, 2).

Jaiswal, Ayush, Rex Yue Wu, Wael Abd-Almageed, and Prem Natarajan (2018). “Unsupervised Adversarial Invariance”. In: *Advances in Neural Information Processing Systems*, pp. 5092–5102 (cit. on pp. 6, 110, 111).

Jaiswal, Ayush, Shuai Xia, Iacopo Masi, and Wael AbdAlmageed (2019). “RoPAD: Robust Presentation Attack Detection through Unsupervised Adversarial Invariance”. In: *arXiv preprint arXiv:1903.03691* (cit. on pp. 81, 110, 111).

Jee, Hyung-Keun, Sung-Uk Jung, and Jang-Hee Yoo (2006). “Liveness Detection for Embedded Face Recognition System”. In: *International Journal of Biological and Medical Sciences* 1.4, pp. 235–238 (cit. on pp. 4, 35).

Karahan, Samil, Merve Kilinc Yildirum, Kadir Kirtac, Ferhat Sukru Rende, Gultekin Butun, and Hazim Kemal Ekenel (2016). “How Image Degradations Affect Deep CNN-Based Face Recognition?” In: *Biometrics Special Interest Group (BIOSIG), 2016 International Conference of The*. IEEE, pp. 1–5 (cit. on p. 63).

Kose, Neslihan and Jean-Luc Dugelay (May 2013). “On the Vulnerability of Face Recognition Systems to Spoofing Mask Attacks”. In: *IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*. Vancouver: IEEE. DOI: http://dx.doi.org/10.1109/ICASSP.2013.6638076 (cit. on pp. 3, 64).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*, pp. 1097–1105 (cit. on pp. 9, 14, 15, 18, 36, 47, 81).

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86.11. 06298, pp. 2278–2324 (cit. on pp. 14, 15, 29).

LeCun, Yann et al. (1989). “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural computation* 1.4, pp. 541–551 (cit. on p. 9).

Li, Haoliang, P. He, S. Wang, A. Rocha, X. Jiang, and A. C. Kot (Oct. 2018). “Learning Generalized Deep Feature Representation for Face Anti-Spoofing”. In: *IEEE Transactions on Information Forensics and Security* 13.10, pp. 2639–2652. DOI: 10.1109/TIFS.2018.2825949 (cit. on pp. 42, 46, 81).

Li, Haoliang, Wen Li, Hong Cao, Shiqi Wang, Feiyue Huang, and Alex C. Kot (2018). “Unsupervised Domain Adaptation for Face Anti-Spoofing”. In: *IEEE Transactions on Information Forensics and Security* 13.7, pp. 1794–1809 (cit. on pp. 43, 46, 47).

Li, L., Z. Xia, A. Hadid, X. Jiang, H. Zhang, and X. Feng (2019). “Replayed Video Attack Detection Based on Motion Blur Analysis”. In: *IEEE Transactions on Information Forensics and Security*, pp. 1–1. DOI: 10.1109/TIFS.2019.2895212 (cit. on p. 81).

Li, Lei, Xiaoyi Feng, Zinelabidine Boulkenafet, Zhaoqiang Xia, Mingming Li, and Abdenour Hadid (2016). “An Original Face Anti-Spoofing Approach Using Partial Convolutional Neural Network”. In: *Image Processing Theory Tools and Applications (IPTA), 2016 6th International Conference On*. IEEE, pp. 1–6 (cit. on p. 81).

Li, Yujia, Kevin Swersky, and Richard Zemel (2014). “Learning Unbiased Features”. In: *arXiv preprint arXiv:1412.5244* (cit. on pp. 6, 110).

Lin, Min, Qiang Chen, and Shuicheng Yan (2013). “Network in Network”. In: *arXiv preprint arXiv:1312.4400* (cit. on pp. 14, 17).

Liu, Ming-Yu and Oncel Tuzel (2016). “Coupled Generative Adversarial Networks”. In: *Advances in Neural Information Processing Systems*, pp. 469–477 (cit. on p. 44).

Liu, Ziwei, Ping Luo, Xiaogang Wang, and Xiaoou Tang (Dec. 2015). “Deep Learning Face Attributes in the Wild”. In: *Proceedings of International Conference on Computer Vision (ICCV)* (cit. on p. 115).

Louizos, Christos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel (2015). “The Variational Fair Autoencoder”. In: *arXiv preprint arXiv:1511.00830* (cit. on pp. 6, 110).

Maaten, Laurens van der and Geoffrey Hinton (2008). “Visualizing Data Using T-SNE”. In: *Journal of machine learning research* 9.Nov, pp. 2579–2605 (cit. on pp. 117, 119).

Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey (2015). “Adversarial Autoencoders”. In: *arXiv preprint arXiv:1511.05644* (cit. on pp. 25, 26).

Mallat, Stéphane (2016). “Understanding Deep Convolutional Networks”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065, p. 20150203 (cit. on p. 96).

Marcel, Sébastien, Mark S. Nixon, Julian Fierrez, and Nicholas Evans, eds. (2019). *Handbook of Biometric Anti-Spoofing: Presentation Attack Detection.* en. 2nd ed. Advances in Computer Vision and Pattern Recognition. Springer International Publishing. DOI: 10.1007/978-3-319-92627-8 (cit. on pp. 3, 34, 82, 109).

McCool, Chris et al. (2012). “Bi-Modal Person Recognition on a Mobile Phone: Using Mobile Phone Data”. In: *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference On.* IEEE, pp. 635–640 (cit. on pp. 59, 65, 127).

Menotti, David et al. (2015). “Deep Representations for Iris, Face, and Fingerprint Spoofing Detection”. In: *IEEE Transactions on Information Forensics and Security* 10.4, pp. 864–879 (cit. on p. 81).

Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida (2018). “Spectral Normalization for Generative Adversarial Networks”. In: *International Conference on Learning Representations* (cit. on p. 115).

Miyato, Takeru, Shin-ichi Maeda, Shin Ishii, and Masanori Koyama (2018). “Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning”. In: *IEEE transactions on pattern analysis and machine intelligence* (cit. on p. 124).

Mohammadi, Amir, Sushil Bhattacharjee, and Sébastien Marcel (2017). “Deeply Vulnerable: A Study of the Robustness of Face Recognition to Presentation Attacks”. In: *IET Biometrics* 7.1, pp. 15–26 (cit. on pp. 3, 63, 125).

Mohammadi, Amir, Sushil Bhattacharjee, and Sébastien Marcel (Jan. 2020a). “Domain Adaptation For Generalization Of Face Presentation Attack Detection In Mobile Settings With Minimal Information”. In: *45th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2020).* IEEE (cit. on p. 125).

Mohammadi, Amir, Sushil Bhattacharjee, and Sébastien Marcel (Jan. 2020b). “Improving Cross-Dataset Performance Of Face Presentation Attack Detection Systems Using Face

Recognition Datasets". In: *45th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2020)*. IEEE (cit. on p. 125).

Nair, Vinod and Geoffrey E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814 (cit. on pp. 15, 18).

Newman, Lily Hay (Aug. 2016). "Hackers Trick Facial-Recognition Logins With Photos From Facebook (What Else?)" en-US. In: *Wired* (cit. on p. 34).

Nikisins, Olegs, Amir Mohammadi, André Anjos, and Sébastien Marcel (2018). "On Effectiveness of Anomaly Detection Approaches against Unseen Presentation Attacks in Face Anti-Spoofing". In: *The 11th IAPR International Conference on Biometrics (ICB 2018)* (cit. on p. 125).

Nosaka, Ryusuke, Yasuhiro Ohkawa, and Kazuhiro Fukui (2011). "Feature Extraction Based on Co-Occurrence of Adjacent Local Binary Patterns". In: *Pacific-Rim Symposium on Image and Video Technology*. Springer, pp. 82–91 (cit. on p. 47).

Pan, Gang, Lin Sun, and Zhaohui Wu (2008). *Liveness Detection for Face Recognition*. INTECH Open Access Publisher (cit. on pp. 4, 35).

Pan, Sinno Jialin and Qiang Yang (2009). "A Survey on Transfer Learning". In: *IEEE Transactions on knowledge and data engineering* 22.10, pp. 1345–1359 (cit. on pp. 5, 40).

Pan, Xingang, Ping Luo, Jianping Shi, and Xiaoou Tang (2018). "Two at Once: Enhancing Learning and Generalization Capacities via Ibn-Net". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 464–479 (cit. on p. 96).

Parkhi, Omkar M., Andrea Vedaldi, and Andrew Zisserman (2015). "Deep Face Recognition". In: *British Machine Vision Conference*. Vol. 1, p. 6 (cit. on pp. 2, 27, 30, 63, 64, 112).

Patel, Keyurkumar, Hu Han, and Anil Jain (2016). "Cross-Database Face Antispoofing with Robust Feature Representation". In: *Chinese Conference on Biometric Recognition*. Springer, pp. 611–619 (cit. on pp. 5, 81).

Patel, V. M., R. Gopalan, R. Li, and R. Chellappa (May 2015). "Visual Domain Adaptation: A Survey of Recent Advances". In: *IEEE Signal Processing Magazine* 32.3, pp. 53–69. DOI: 10.1109/MSP.2014.2347059 (cit. on pp. 42, 94).

Pereira, T. de Freitas, André Anjos, and Sébastien Marcel (July 2019). "Heterogeneous Face Recognition Using Domain Specific Units". In: *IEEE Transactions on Information Forensics and Security* 14.7, pp. 1803–1816. DOI: 10.1109/TIFS.2018.2885284 (cit. on pp. 82, 83, 85, 96, 100, 106).

Platt, John et al. (1999). "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods". In: *Advances in large margin classifiers* 10.3, pp. 61–74 (cit. on p. 76).

Prince, Simon JD and James H. Elder (2007). "Probabilistic Linear Discriminant Analysis for Inferences about Identity". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference On*. IEEE, pp. 1–8 (cit. on p. 27).

Quionero-Candela, Joaquin, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence (2009). *Dataset Shift in Machine Learning*. The MIT Press (cit. on pp. 5, 40).

Ramachandra, Raghavendra et al. (Dec. 2019). “Smartphone Multi-Modal Biometric Authentication: Database and Evaluation”. In: *arXiv:1912.02487 [cs]* (cit. on pp. 59, 114, 126).

Reynolds, Douglas A, Thomas F Quatieri, and Robert B Dunn (2000). “Speaker Verification Using Adapted Gaussian Mixture Models”. In: *Digital signal processing* 10.1, pp. 19–41 (cit. on pp. 32, 112).

Russakovsky, Olga et al. (2015). “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y (cit. on pp. 9, 14).

Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton (2017). “Dynamic Routing between Capsules”. In: *Advances in Neural Information Processing Systems*, pp. 3856–3866 (cit. on p. 124).

Sandberg, David (2017). “Facenet: Face Recognition Using Tensorflow”. In: (cit. on pp. 30, 31, 64, 112).

Sanger, Terence D. (1989). “An Optimality Principle for Unsupervised Learning”. In: *Advances in Neural Information Processing Systems*, pp. 11–19 (cit. on p. 15).

Scherhag, Ulrich, R. Raghavendra, K. B. Raja, M. Gomez-Barrero, C. Rathgeb, and C. Busch (2017). “On the Vulnerability of Face Recognition Systems towards Morphed Face Attacks”. In: *Biometrics and Forensics (IWBF), 2017 5th International Workshop On*. IEEE, pp. 1–6 (cit. on p. 64).

Schroff, Florian, Dmitry Kalenichenko, and James Philbin (2015). “Facenet: A Unified Embedding for Face Recognition and Clustering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 00297, pp. 815–823 (cit. on pp. 2, 27, 30, 31, 63).

Shao, Rui, Xiangyuan Lan, Jiawei Li, and Pong C. Yuen (June 2019). “Multi-Adversarial Discriminative Deep Domain Generalization for Face Presentation Attack Detection”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 43).

Shor, Joel et al. (2019). “Personalizing ASR for Dysarthric and Accented Speech with Limited Data”. In: *arXiv preprint arXiv:1907.13511* (cit. on pp. 96, 100, 106).

Simonyan, Karen and Andrew Zisserman (Sept. 2014). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv:1409.1556 [cs]* (cit. on pp. 15, 16).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *The journal of machine learning research* 15.1, pp. 1929–1958 (cit. on p. 14).

Storkey, Amos (2009). “When Training and Test Sets Are Different: Characterizing Learning Transfer”. In: *Dataset shift in machine learning*, pp. 3–28 (cit. on p. 40).

Sun, Yi, Ding Liang, Xiaogang Wang, and Xiaoou Tang (2015). “DeepID3: Face Recognition with Very Deep Neural Networks”. In: *arXiv preprint arXiv:1502.00873* (cit. on pp. 2, 27, 30, 63).

Szegedy, Christian, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi (Feb. 2016). “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: *arXiv:1602.07261 [cs]* (cit. on pp. 20, 21, 23, 31, 82, 85).

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna (2016). "Rethinking the Inception Architecture for Computer Vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826 (cit. on pp. 20, 21).

Szegedy, Christian et al. (2015). "Going Deeper with Convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 01905, pp. 1–9 (cit. on pp. 17, 18, 20, 21, 129).

Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf (2014). "Deepface: Closing the Gap to Human-Level Performance in Face Verification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1701–1708 (cit. on pp. 2, 27, 29, 63).

Trusted Biometrics under Spoofing Attacks (TABULA RASA) (2016). http://www.tabularasa-euproject.org/ (cit. on p. 63).

Tu, Xiaoguang, Hengsheng Zhang, Mei Xie, Yao Luo, Yuefei Zhang, and Zheng Ma (2019). "Deep Transfer Across Domains for Face Anti-Spoofing". In: *arXiv preprint arXiv:1901.05633* (cit. on p. 110).

Tu, Xiaoguang, Jian Zhao, et al. (2019). "Learning Generalizable and Identity-Discriminative Representations for Face Anti-Spoofing". In: *arXiv preprint arXiv:1901.05602* (cit. on pp. 43, 81, 96, 110, 111, 124).

Tu, Xiaokang and Yuchun Fang (2017). "Ultra-Deep Neural Network for Face Anti-Spoofing". In: *International Conference on Neural Information Processing*. Springer, pp. 686–695 (cit. on p. 81).

Turk, Matthew and Alex Pentland (1991). "Eigenfaces for Recognition". In: *Journal of cognitive neuroscience* 3.1, pp. 71–86 (cit. on pp. 2, 27).

Tzeng, Eric, Judy Hoffman, Trevor Darrell, and Kate Saenko (2015). "Simultaneous Deep Transfer across Domains and Tasks". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4068–4076 (cit. on p. 44).

Tzeng, Eric, Judy Hoffman, Kate Saenko, and Trevor Darrell (2017). "Adversarial Discriminative Domain Adaptation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7167–7176 (cit. on pp. 44–46).

Vogt, Robbie and Sridha Sridharan (2008). "Explicit Modelling of Session Variability for Speaker Verification". In: *Computer Speech & Language* 22.1, pp. 17–38 (cit. on pp. 33, 112).

Wallace, Roy, Mitchell McLaren, Christopher McCool, and Sébastien Marcel (2011). "Inter-Session Variability Modelling and Joint Factor Analysis for Face Authentication". In: *Biometrics (IJCB), 2011 International Joint Conference On*. IEEE, pp. 1–8 (cit. on pp. 27, 31–33, 64, 112).

Wang, Mei and Weihong Deng (2018). "Deep Visual Domain Adaptation: A Survey". In: *Neurocomputing* 312, pp. 135–153 (cit. on pp. 5, 42, 94).

Watson, Craig I. (May 2010). *Multiple Encounter Dataset I (MEDS-I)*. NIST Pubs 7679. NIST, p. 16 (cit. on p. 33).

Wen, D., H. Han, and Anil Jain (Apr. 2015). "Face Spoof Detection With Image Distortion Analysis". In: *IEEE Transactions on Information Forensics and Security* 10.4, pp. 746–761. DOI: 10.1109/TIFS.2015.2400395 (cit. on pp. 4, 35, 46, 47, 59, 60, 65, 67, 127).

Wiskott, Laurenz, Jean-Marc Fellous, N. Kuiger, and Christoph Von Der Malsburg (1997). "Face Recognition by Elastic Bunch Graph Matching". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19.7, pp. 775–779 (cit. on pp. 2, 27).

Wu, Xiang, Ran He, Zhenan Sun, and Tieniu Tan (2015). "A Light CNN for Deep Face Representation with Noisy Labels". In: *arXiv preprint arXiv:1511.02683* (cit. on pp. 30, 31, 64, 112).

Xie, Qizhe, Zihang Dai, Yulun Du, Eduard Hovy, and Graham Neubig (2017). "Controllable Invariance through Adversarial Feature Learning". In: *Advances in Neural Information Processing Systems*, pp. 585–596 (cit. on pp. 6, 110).

Xu, Zhenqi, Shan Li, and Weihong Deng (2015). "Learning Temporal Features Using LSTM-CNN Architecture for Face Anti-Spoofing". In: *Pattern Recognition (Acpr), 2015 3rd IAPR Asian Conference On.* IEEE, pp. 141–145 (cit. on p. 81).

Yang, Jianwei, Zhen Lei, and Stan Z. Li (Aug. 2014). "Learn Convolutional Neural Network for Face Anti-Spoofing". In: *arXiv:1408.5601 [cs]* (cit. on pp. 5, 36, 37, 81).

Ying, Xiaowen, Xin Li, and Mooi Choo Chuah (2018). "LiveFace: A Multi-Task CNN for Fast Face-Authentication". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA).* IEEE, pp. 955–960 (cit. on pp. 81, 111, 124).

Yosinski, Jason, Jeff Clune, Yoshua Bengio, and Hod Lipson (2014). "How Transferable Are Features in Deep Neural Networks?" In: *Advances in Neural Information Processing Systems*, pp. 3320–3328 (cit. on p. 96).

Zeiler, Matthew D. and Rob Fergus (2014). "Visualizing and Understanding Convolutional Networks". In: *European Conference on Computer Vision.* Springer, pp. 818–833 (cit. on p. 96).

Zhang, Kaipeng, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao (2016). "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks". In: *IEEE Signal Processing Letters* 23.10, pp. 1499–1503 (cit. on p. 29).

Zhang, Zhiwei, Junjie Yan, Sifei Liu, Zhen Lei, Dong Yi, and Stan Z. Li (2012). "A Face Anti-spoofing Database with Diverse Attacks". In: *Biometrics (ICB), 2012 5th IAPR International Conference On.* IEEE, pp. 26–31 (cit. on pp. 46, 47).

Zhang, Zhiwei, Dong Yi, Zhen Lei, and S.Z. Li (Mar. 2011). "Face Liveness Detection by Learning Multispectral Reflectance Distributions". In: *2011 IEEE International Conference on Automatic Face Gesture Recognition and Workshops (FG 2011)*, pp. 436–441. DOI: 10.1109/FG.2011.5771438 (cit. on p. 4).

Zhao, Shengjia, Jiaming Song, and Stefano Ermon (June 2017). "InfoVAE: Information Maximizing Variational Autoencoders". In: *arXiv:1706.02262 [cs, stat]* (cit. on pp. 25, 113).

Zhou, F. et al. (July 2019). "Face Anti-Spoofing Based on Multi-Layer Domain Adaptation". In: *2019 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pp. 192–197. DOI: 10.1109/ICMEW.2019.00-88 (cit. on p. 43).

Index

Amir Mohammadi

Curriculum Vitae

1920 Martigny, Switzerland
+41 77 951 6981
amir.mohammadi@bluewin.ch
linkedin.com/in/amirmohammadi

Education

2006–2011	**B.Sc. in Electrical Engineering, Bioelectric**, *University of Tehran*, Tehran, Iran.
2011–2014	**M.Sc. in Electrical and Electronics Engineering**, *Özyeğin University*, Istanbul, Turkey.
2016–Jan 2020	**Ph.D. in Electrical Engineering**, *École Polytechnique Fédérale de Lausanne (EPFL)*, Switzerland.

Experience

Feb 2016 – Jan 2020 **Research Assistant**, IDIAP, Switzerland, under supervision of Dr. Sébastien Marcel.
Worked on **face recognition**, **speaker recognition**, and their anti-spoofing.
- Was the **responsible person for the project** that funded the PhD. Tracked and completed all the deliverables on time.
- **Designed a data collection and collected biometric data** from 60 participants in six sessions over a period of four months. Also, **supervised an intern** for another data collection.
- **Developed deep learning systems** for face anti-spoofing.
- **Became a core developer** of Bob: a multi-package signal processing and machine learning toolbox. **Helped build a sophisticated DevOps** for Bob.
- Became a **machine learning expert** by taking several machine learning courses and applying them on real problems.

Mar 2015 – Aug 2015 **Internship**, NATIONAL INSTITUTE OF INFORMATICS, Japan.
Worked on liveness detection of speech signals; collected data as well.

Oct 2011 – Sep 2014 **Research Assistant**, SPEECH LAB, Özyeğin University, Turkey.
Worked on text to speech synthesis systems. Developed a Python library in our team which everyone uses to conduct their experiments.

Jun 2010 – Sep 2010 **Internship**, KARA ELECTRONICS, Tehran, Iran.
Learned some practical work and designed a solar powered outdoor lamp and an RGB LED driver circuit. The biggest experience was to complete the whole work from beginning to the end with little guidance of my supervisor.

Publications

Please see the full list on: `https://scholar.google.com/citations?user=lv3UX84AAAAJ`

Course Projects

July 2017 **Fundamentals in Statistical Pattern Recognition**, *Dr. S. Marcel and Dr A. Anjos*, EPFL.
Completed projects on linear regression, logistic regression, neural networks, PCA, LDA, K-Means, GMMs, and SVMs. Implemented an LDA-based face recognition system for the final project.

July 2016	**Image analysis and pattern recognition**, *Prof. Jean-Philippe THIRAN*, EPFL. Completed two labs on image segmentation and recognition and a final project where we had to guide a robot to follow arrows on an arena using a camera.
Dec 2013	**Several Projects on Digital Image Processing**, DIGITAL IMAGE PROCESSING COURSE PROJECTS, Prof. Tanju Erdem, Özyeğin University. I have done several course projects on Digital Image Processing including Image Enhancement, Resampling, Edge Detection, Fourier Transform, Image Warping, etc. using MATLAB.
May 2013	**Computer Vision and Digital Image Processing**, *Prof. Tanju Erdem*, Özyeğin University. Completed several projects in computer vision and image processing including Camera Calibration, Stereo Vision, Feature Detection, Panorama Creation, 3D reconstruction, Image Enhancement, Resampling, Edge Detection, Fourier Transform, and Image Warping using OpenCV and MATLAB.

Skills & Expertise

Programming	PYTHON, C/C++, MATLAB
Toolboxes	Tensorflow, PyTorch, Keras, Scikit-learn, OpenCV, Kaldi
Miscellaneous	GNU/Linux, Shell scripting, distributed computing, virtualization, packaging
Languages	Persian (native), English (fluent, C1), French (intermediate, A2-B1)

Personal Interests

- Biking with friends in Switzerland
- Cooking and trying new recipes