OPEN SOURCE PRESENTATION ATTACK DETECTION BASELINE FOR IRIS RECOGNITION

An Undergraduate Thesis

Submitted to the College of Engineering
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Science
in
Electrical Engineering

by
Joseph G. McGrath

Adam Czajka, Director

Bachelor's Program in Electrical Engineering
Notre Dame, Indiana
April 2019

This document is in the public domain.

OPEN SOURCE PRESENTATION ATTACK DETECTION BASELINE FOR IRIS RECOGNITION

Abstract

by

Joseph G. McGrath

This thesis proposes the first, known to us, open source presentation attack detection (PAD) solution to distinguish between authentic irises (possibly wearing clear contact lenses) and irises with textured contact lenses. This software can serve as a baseline in various PAD evaluations, and also as an open-source platform with an up-to-date reference method for iris PAD. The software is written in C++ and Python and uses only open source resources, such as OpenCV. This method does not incorporate iris image segmentation, which may be problematic for unknown fake samples. Instead, it makes a best guess to localize the approximate position of the iris. The PAD-related features are extracted with the Binary Statistical Image Features (BSIF), which are classified by an ensemble of classifiers incorporating support vector machine, random forest, and multilayer perceptron. The models attached to the current release have been trained with the NDCLD'15 database and evaluated on the independent datasets included in the LivDet-Iris 2017 competition. The software implements the functionality of retraining the classifiers with any database of authentic and attack images. The accuracy of the current version offered with this thesis exceeds 99% when tested on subject-disjoint subsets of NDCLD'15 and oscillates around 85% when tested on the LivDet-Iris 2017 benchmarks, which is on par with the results obtained by the LivDet-Iris 2017 winner.

CONTENTS

FIGURES

TABLES

CHAPTER 1

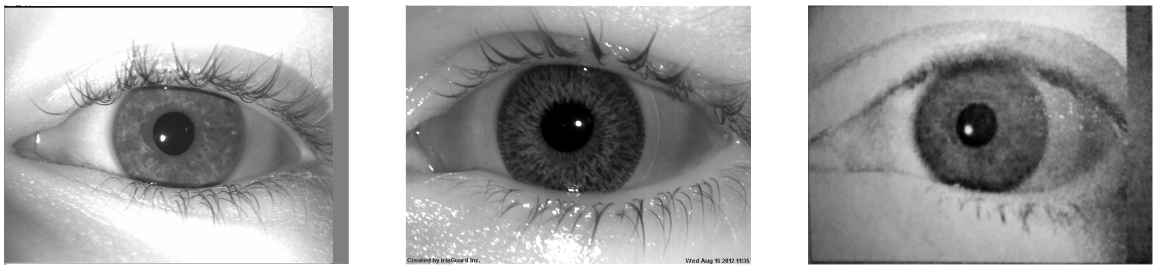
INTRODUCTION

1.1 Overview

Biometric security systems are designed to recognize individuals on the basis of biological characteristics – such as iris texture – that are unique to each individual. The goal of such a system in general is to provide a recommendation for future action based upon the identity of an individual. In the case of a friendly individual, the system will allow the user to gain access, while in the case of a threatening individual, the system will not allow the user to gain access and may activate a warning system.

Iris texture is a commonly used biometric for security systems. The iris has many advantages over other biometrics; for example, it has a multi-dimensional structure that allows for more variation between individuals than the single dimensional structure of biometrics such as fingerprints. The pattern formed in the iris depends on the unique interplay between a multitude of features such as furrows, crypts, and rings. Matching is performed through a test of statistical independence based on the features present in the iris. If a set of iris features fails the test of statistical independence, a match has been detected [7].

Current deployments of iris recognition systems include a port of entry recognition system in the UAE and the Aadhaar biometric identification system in India. Such systems are becoming ubiquitous as they appear in smartphones such as the Samsung Galaxy S8 and other applications such as ATMs.



(a) Live iris (b) Textured contact lens (c) Printout

Figure 1.1. Example iris presentations

1.2 Motivation

1.2.1 Presentation Attack Detection

A threatening user may desire to instead be recognized as a friendly or unknown individual. This individual will undertake a presentation attack, which is to make a presentation to the biometric system in an attempt to manipulate the system into an incorrect decision. An imposter attack presentation is when the malicious individual attempts to be recognized by the system as a different, friendly individual, while a concealer attack presentation is when the individual merely desires to obscure the useful biometric information. Presentation attack detection (PAD) is therefore a vital component of any biometric security system because it allows the system to be robust to malicious attacks.

In the context of iris recognition systems, PAD can involve the detection of a multitude of presentation attack instruments (PAI) such as paper printouts, cadaver eyes, and textured contact lenses. Example presentations to an iris system can be found in Fig. 1.1. Iris PAD is not a solved problem: numerous examples can be found demonstrating successful presentation attacks on commercially available systems, suggesting PAD mechanisms were either ineffective or missing in these systems. In 2002, paper printouts of an iris were used to match with an identity registered with Panasonic's Authenticam BM-ET100 [21] and more recently paper printouts and contact lenses were used to spoof the iris scanner on Samsung's Galaxy S8[1]. If high quality iris information can be obtained from personal photographs, it could be combined with these spoofing methods to successfully access any biometric system where the individual is registered [22]. Without successful iris PAD, security systems that utilize the uniqueness of individual iris textures would no longer be secure; therefore, iris PAD is critical.

In this thesis, the primary focus will be the detection of textured contact lenses as they are one of the simplest methods for concealer attack presentation and may be less obvious to human components in the security system; however, the software presented in this thesis is able to train models that may be specialized to other PAI. Textured contact lenses are manufactured with the intent to modify the appearance of the wearer's eye by changing the texture or color. Therefore, textured contact lenses significantly reduce the visible live iris texture and can allow threatening users to conceal their identities. It has been shown that textured contact lenses are able to lower the verification rate of users to below 40% [26].

1.2.2 Open Source Baseline

One possible reason for slow progress in any field is the lack of a freely available foundation of research to build upon. This is especially true in computer science, as many solutions involve software components that have been implemented previously. If an open source platform is available to build upon, researchers can focus on merely modifying the existing code base to accommodate a novel proposal rather than creating an entirely new implementation.

An open source platform to maintain a baseline iris PAD methodology would increase the rate of progression of iris PAD research. Such a platform would be

[1]`https://goo.gl/zjEF3M`, The Guardian, May 2017

easy to contribute to and easy to benefit from when developing or evaluating original solutions. The OpenCV platform[2] is a great example of such an initiative in computer vision in general. Through OpenCV, researchers are able to focus on innovative ideas and the production of solutions with increased readability, which allows for even more innovation [1]. The Masek implementation and more recently the OSIRIS system have played a similar role for iris recognition. OSIRIS has served a dual purpose within the research community, both providing an open source platform that can be modified to improve performance and a baseline iris recognition solution that can be used to benchmark the performance of other algorithms [17].

The success of other open source computer vision platforms has motivated the development of this baseline, which can be accessed through its GitHub repository at `https://github.com/CVRL/OpenSourceIrisPAD`. This open source platform provides the research community with a starting point for iris PAD, which is vital for the continued security of iris texture based biometric security systems. In addition to the software available, ready to use models that have been trained on NDCLD'15[3] have been provided at `https://notredame.box.com/v/OpenSourceIrisPADModels`. A condensed version of this thesis has been submitted to the Biometrics: Theory, Applications and Systems (BTAS) 2019 conference[4]. A preprint of this submission is available on arXiv at `https://arxiv.org/abs/1809.10172`.

1.3 Standardization

Various standards for iris PAD and biometrics in general have been introduced by the International Organization for Standardization (ISO) and the International

[2]`https://opencv.org`

[3]`https://cvrl.nd.edu/projects/data/#the-notre-dame-contact-lense-dataset-2015ndcld15`

[4]`http://ieee-biometrics.org/btas2019/`

Electrotechnical Commission (IEC). ISO/IEC 30107 provides the standard for PAD where the attack occurs at the sensor rather than at another point in the biometric system. The framework and vocabulary defined in the freely available ISO/IEC 30107-1[5], such as PAD and PAI, will be used throughout this paper.

The metrics defined in ISO/IEC 30107-3 will be used to evaluate the baseline open source iris PAD solution. These include the attack presentation classification error rate (APCER) and the bona fide presentation classification error rate (BPCER). The APCER is the proportion of attack presentations incorrectly classified as bona fide presentations while the BPCER is the proportion of bona fide presentation incorrectly classified as attack presentations.

1.4 Related Work

Iris PAD is a dynamic research area with many different methods proposed to date. A recent survey by Czajka and Bowyer classifies the various PAD methods into four separate categories, based on two separate labels: a method may be *static* or *dynamic* and also *passive* or *active* [4]. *Static* methods use still images while *dynamic* methods use video to analyze the changing features of the iris, such as the pupil size. A method is *passive* if no additional stimulation of the eye is performed and *active* otherwise. Various PAI require different methods as they exploit the biometric security system in different ways.

The method used in this paper to perform iris PAD will be *static* and *passive* because of the focus on textured lens detection. Static-passive PAD methods typically rely on classifiers that are trained to recognize presentation attacks on the basis of various hand-crafted texture descriptors. One such texture descriptor is Local Binary Pattern (LBP), which was used to achieve high accuracy recognition of textured

[5]`https://goo.gl/JSbiqy`

contact lenses on the Notre Dame Cosmetic Contact Lens 2012 dataset [10]. However, the performance significantly decreases when novel textured lens types are considered in the testing set [9]. Another texture descriptor, binarized statistical image features (BSIF), has been shown to outperform LBP in generalization tests across texture contact lens brands [14], a result that is confirmed and extended to multiple iris segmentation techniques and a variety of model types by Doyle *et al.* [8].

A more recent approach to static-passive iris PAD has been through the use of convolutional neural networks [16]. These networks are able to automate the process of learning image features that are relevant to iris PAD; thus, systems employing this type of iris PAD are able to avoid the potential limitations of hand-crafted features. Combined iris localization and PAD has been explored as well and gives state of the art results without requiring iris segmentation [2].

For other categories of PAD – not static-passive – many different approaches exist. If some modifications in the iris recognition equipment are possible, iris PAD methods incorporate multi-spectral imaging solely in near-infrared band [18] or combined with visible-light imaging [22]; 3D properties of the eye [5, 15]; or dynamic features such as spontaneous [24] or stimulated [3] pupil oscillations, eye blinks [20], or eyeball movements [13].

In the context of the existing tools for and efforts toward faster development of iris PAD methodologies, it is worth mentioning numerous benchmark databases, such as Clarkson, Warsaw, Notre Dame, and WVU/IIITD-Delhi developed for LivDet-Iris competitions [27–29] (paper printouts and textured contact lenses); NDCCL 2012 [10], NDCLD 2013, [9] and NDCLD 2015 [8] (clear and textured contact lenses); ATVS-FIr [11] (paper printouts); Pupil-Dynamics [3] (pupil size in time with and without visible-light stimuli); Post-Mortem-Iris [23] (images of irises acquired up to one month after death); CASIA-Iris-Syn [25] (synthetically generated iris images); or data acquired by a LightField sensor GUC-LF-VIAr-DB [19].

The LivDet-Iris competitions[6] are an important effort towards independent evaluation of iris PAD algorithms. Editions were organized in 2013 [27], 2015 [29], and 2017 [28] and brought together researchers from around the world, who submitted their iris PAD algorithms for evaluation. In the most recent edition, LivDet-Iris 2017 [28], the winning algorithm was unable to recognize between 11% and 38% of the attack images in the open-set regime, where some (or all) properties of the testing samples are unknown during training. Thus, despite the multitude of solutions proposed so far, there is still a significant amount of progress to be made in iris PAD.

[6]`http://livdet.org`

CHAPTER 2

BASELINE METHOD FOR TEXTURED CONTACT LENS DETECTION

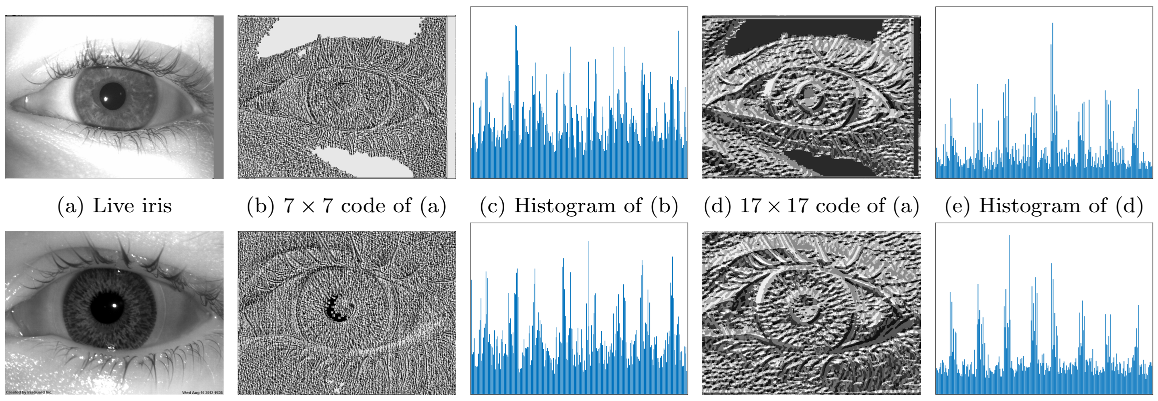
The implemented solution follows the methodology proposed by Doyle and Bowyer [8] and the feature extraction is based on Binary Statistical Image Features (BSIF) proposed by Kannala and Rahtu [12]. This method was chosen to serve as a baseline for open source iris PAD because it is one of the more recent and effective methods.

2.1 BSIF Image Features

In this method, the calculated "BSIF code" is based on filtering the image with n filters of size $s \times s$ and then binarizing the filtering results with a threshold at zero. Hence, for each pixel n binary responses are given, which are in the next step translated into a n-bit grayscale value. Fig. 2.1 presents BSIF codes for example iris images (with and without textured contact lenses) for two example scales ($s = 7$ and $s = 17$) and $n = 8$.

The histograms resulting from gray-scale BSIF codes are later normalized to a z-score and used as texture descriptors with the number of histogram bins equal to 2^n. In addition to the original ISO-compliant iris image resolution of 640×480, BSIF codes for an image down-sampled to 320×240 were extracted. This allowed for the exploration of more scales in feature extraction as this effectively doubles the filter scale.

The *Best Guess* segmentation technique explored by Doyle and Bowyer has been implemented as standard in the baseline method. For *Best Guess* segmentation, a region of interest is selected that corresponds to the average iris center point and


(a) Live iris (b) 7×7 code of (a) (c) Histogram of (b) (d) 17×17 code of (a) (e) Histogram of (d)

(f) Textured contact (g) 7×7 code of (f) (h) Histogram of (g) (i) 17×17 code of (f) (j) Histogram of (i)

Figure 2.1. 8-bit BSIF codes and the resulting histograms calculated at two example scales for an authentic iris image and an iris image with textured contact lens.

radius from the training set. For the ISO-compliant iris images in NDCLD'15, this corresponds to a center location of $(320, 250)$ with a radius of 125 pixels.

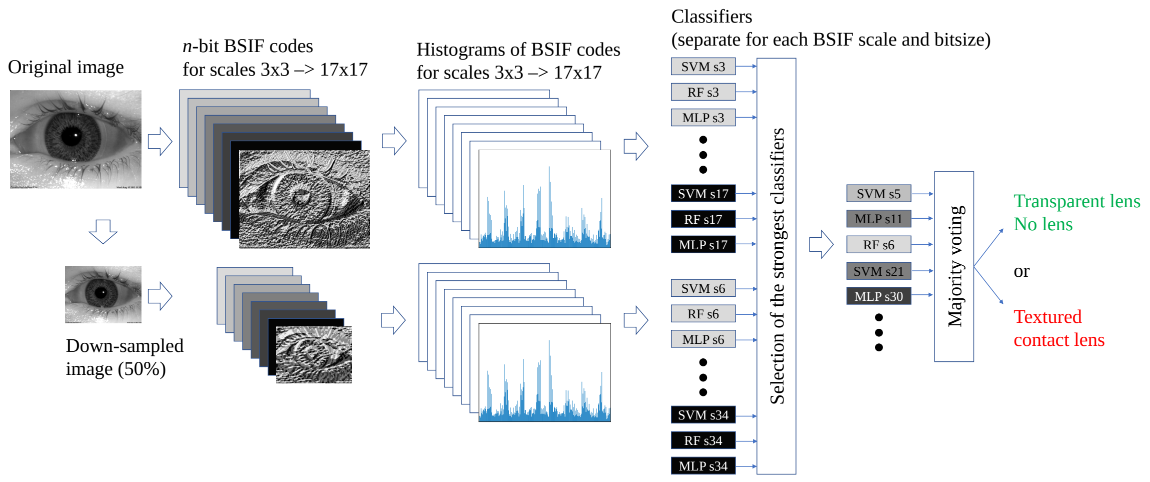


Figure 2.2: A schematic view of the proposed open-source iris PAD solution.

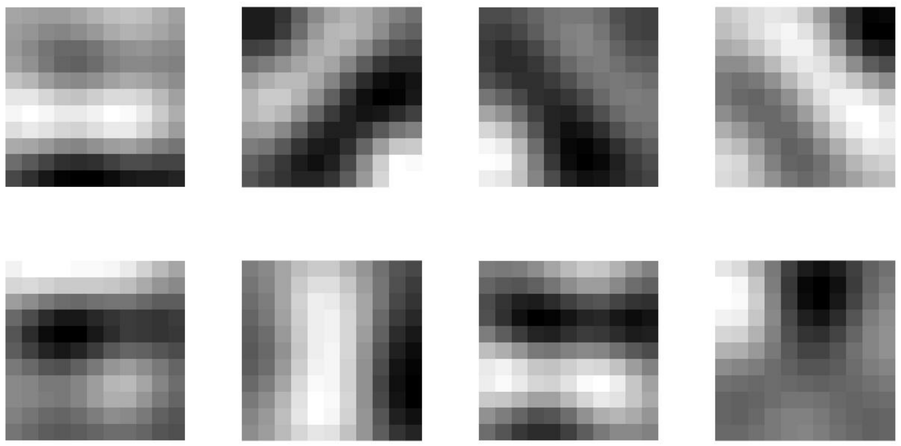


Figure 2.3. Example filters with $n = 8$ and $s = 11$.

2.1.1 Default BSIF

In the original BSIF paper by Kannala and Rahtu, $n \in \{5, 6, 7, 8, 9, 10, 11, 12\}$ and $s \in \{3, 5, 7, 9, 11, 13, 15, 17\}$; thus, there are 60 combinations of n and s (4 combinations, namely $n \in \{9, 10, 11, 12\}$ for $s = 3$, were not available). When the images are down-sampled to 320×240, the number of combinations can be doubled to 120 through the effective addition of $s \in \{6, 10, 14, 18, 22, 26, 30, 34\}$. The filters, for each considered combination of n and s, were trained on patches extracted from natural images in a way to maximize the statistical independence of filter responses. An example of the filters used for $n = 8$ and $s = 11$ can be seen in Fig. 2.3.

To extend the original methodology, in which only $n = 8$ filters were used, the method has been implemented here for all n as proposed in the original BSIF paper. The method is equivalent for $n \neq 8$, the only difference being the length of each BSIF code. This addition allows for more options when searching for the optimal ensemble of classifiers.

2.1.2 Domain Specific BSIF

An additional extension of the method proposed is the inclusion of domain-specific BSIF filters, which have been shown to out-perform the default, natural image BSIF filters [6]. The filters used were trained on gaze-based regions of interest from iris images in such a way as to maximize statistical independence, just as was done in default BSIF. For the domain-specific filters, $n \in \{5, 6, 7, 8, 9, 10, 11, 12\}$ and $s \in \{5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 33, 39\}$, giving 96 possible combinations of n and s. With down-sampling, the number of combinations can be doubled to 192.

2.2 Models

Three different model types were included in the baseline method to provide a wide range of possibilities, especially when combined with the large number of possible BSIF feature sets. The models chosen for inclusion in the method are support vector machine (SVM), random forest, and multilayer perceptron. A separate set of models can be trained for each feature set, giving 360 models total for standard BSIF and 576 models total for domain specific BSIF.

The classifiers will vary in strength on the validation set and therefore a subset of the larger set of models is selected for use on the testing set. To select this subset, the models are ranked by their individual performance on the validation set. These models are then added one by one to an ensemble of classifiers to use in majority voting. The number of models that produces the maximum performance on the validation set is taken as the optimal number of models. The selected ensemble can then be used for majority voting on other datasets or directly for iris PAD in biometric security systems.

CHAPTER 3

SOFTWARE ARCHITECTURE

The `TCL Detection` solution proposed in this paper supports textured contact lens (TCL) detection and includes three main modes of operation: feature extraction, model training, and model testing.

3.1 Tools Used

3.1.1 Languages

The `TCL Detection` solution includes versions written in C++ and Python. C++ was chosen initially due to performance considerations: while security is the main goal of an iris recognition system, the system should also be reasonably fast. Another consideration when C++ was chosen was compatibility with OSIRIS for the incorporation of automated iris segmentation in the future.

`TCL Detection` was also implemented in Python to support ease of understanding. The baseline will be able to foster further research in iris PAD without requiring users to allocate large amounts of time to understanding the implementation.

Both versions have been included to ensure that there is an open source baseline that can be compared to modern commercial systems where performance is important while also providing an easy to use system for researchers to build upon.

3.1.2 Libraries

OpenCV version 3.4.1 was used for its image handling and implementation of machine learning models and training functions. This library was chosen because it is well established in the field of computer vision and is considered the state of the art in open source computer vision implementations.

HDF5 version 1.10.4 was also used to store extracted features in the HDF5 file format, which was chosen due to its speed and precision.

3.1.3 Compilation

`TCL Detection` was tested on MacOS High Sierra 10.13.6 using g++ 4.2.1 in addition to Xcode 10.1 and on Windows 7 64 bit using Microsoft Visual Studio 2015. A `makefile` is provided to assist with the compilation of the C++ version. This `makefile` uses the helper tool pkg-config to ensure that the OpenCV library is correctly included.

3.2 Functionality

`TCL Detection` was designed to provide a baseline open source implementation for iris PAD. As such, it can extract BSIF features, train various models, and test these models.

Feature extraction will work with filters of any scale and with any number of bits. The implementation has been tested with all filters contained in standard BSIF as well as domain specific filters created for iris recognition. Additionally, the filter sizes can be effectively doubled by down-sampling the images prior to filtering. The results of filtering each image are summarized in a histogram, which is output to a file for future use.

The training portion of the program is capable of training three different types of machine learning models – support vector machine (SVM), random forest, and multilayer perceptron. Hyperparameter optimization is performed for each model and specific results will be discussed in chapter 4. After training, the models are saved as `xml` files.

Testing consists of two different modes: single model and majority voting. In the single model mode, the performance of each model on the testing set is determined separately. If majority voting is enabled, an ensemble of models will be used to vote on the classification of each image. In the case of a tie, a random assignment will be made.

3.3 Class Structure

The overall structure of both versions (C++ and Python) of `TCL Detection` is based on the structure of OSIRIS version 4.1 in that a manager class is used to control all information flow within the application.

The operation of the program is controlled by changing settings in an external configuration file. One of the primary responsibilities of the `TCLManager` class is loading the configuration for the current iteration of the program using the `loadConfig()` function. This function loads the operation commands as well as other various parameters such as the directories, images, and models to use. The `TCLManager` class also contains a function `showConfig()` to display the current selections to the user. The `run()` function can then be called to start the main program tasks. The manager achieves this by creating instances of the other classes necessary to the operation of the program.

The `featureExtractor` class (`filter` module in Python) handles the extraction of BSIF features for a given list of image files. The `extract()` method of the `featureExtractor` and `filter` module can be used to extract and save BSIF features for a specific scale s and number of bits n. This method creates an instance of the BSIF filter that is then used to filter a regular or down-sampled version of each image, depending on the input scale. The `featureExtractor` class and `filter` module also serve as the interface with the required HDF5 functions: they create a new file for each combination of s, n and place the feature sets within the file, indexed by the name of the image they represent.

The `BSIFFilter` class contains the methods for loading the hard-coded BSIF filters and for generating a histogram for an image using the method described in chapter 2. In the Python implementation, the C++ version of the `BSIFFilter` class is used with Python bindings to load the required filters. The `generateHistogram()` method assigns a bit to each filter used, giving an n-bit integer for each pixel corresponding to the response of the filters. A histogram is then taken across the entire image and returned to the `extract` method, which saves the histogram as the feature set for that image, bit size, and filter scale.

The remainder of the operation modes are handled by the manager class, which instantiates the required OpenCV objects to train models and test images. If model training is enabled, the manager will call a method, `loadFeatures`, to load the features for the images specified in the training list. The training features and classifications are then loaded into an instance of the `TrainData` class from OpenCV for C++ or a NumPy array for Python. A new model of the type specified in the configuration file is then initialized and trained; currently, the supported model types are SVM, random forest, and multilayer perceptron. Training is achieved through the `trainAuto` function in OpenCV for SVM and through custom training functions designed to replicate the function of `trainAuto` for random forest and multilayer perceptron. These training functions choose the optimal parameters using k-fold cross validation with ten folds. Each model that is trained is then output as an `xml` file.

If model testing is enabled, the manager will load the required models from their `xml` files. If majority voting is disabled, the manager will individually load each model and the testing features corresponding to the BSIF scale and bit size the model was trained on. These features will then be input to the `predict` function for the model from OpenCV and the predictions will be returned. The predictions will be tested against the classifications provided with the test set and the CCR, APCER, and

BPCER will be output for each model individually.

If majority voting is enabled, the predictions for each model are determined and temporarily stored. For each image, the number of models voting for each classification is determined and the overall decision is made with a simple majority vote. In the case of a tie, a random decision is made. The ensemble accuracy on the training set is then determined through comparison with the classifications provided with the test set.

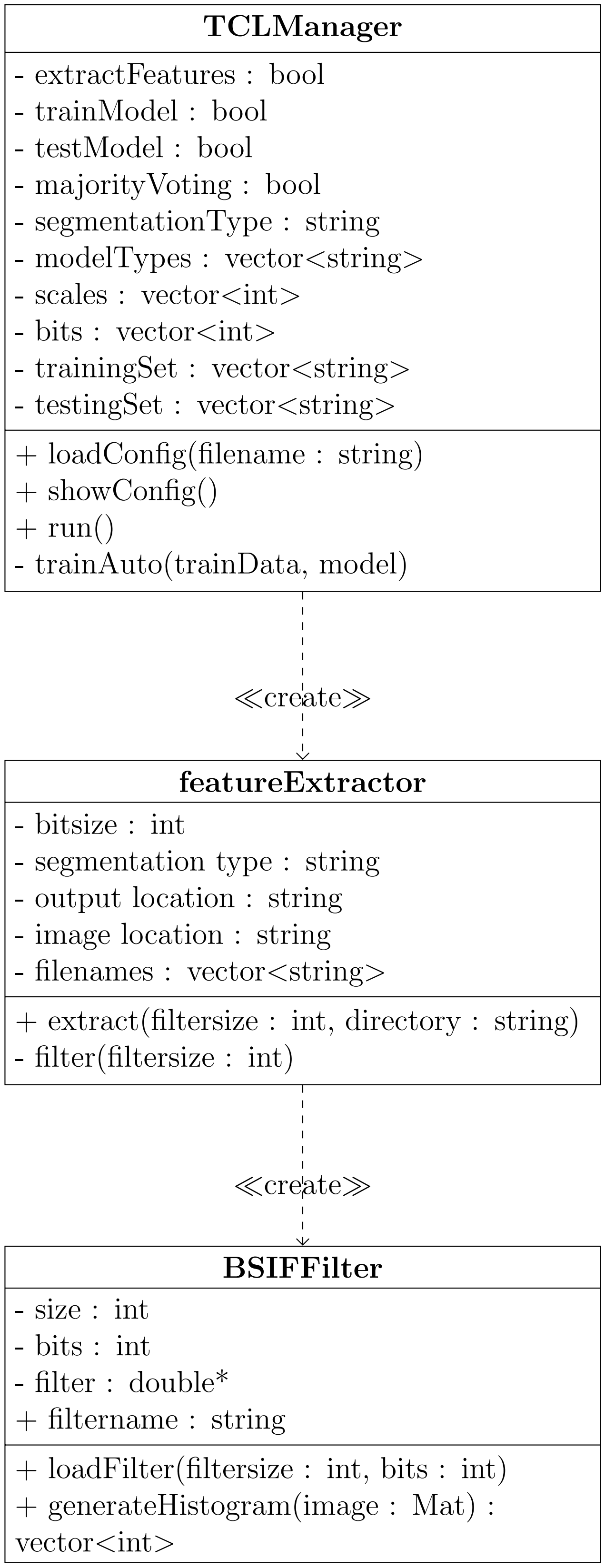

TCLManager
- extractFeatures : bool
- trainModel : bool
- testModel : bool
- majorityVoting : bool
- segmentationType : string
- modelTypes : vector<string>
- scales : vector<int>
- bits : vector<int>
- trainingSet : vector<string>
- testingSet : vector<string>
+ loadConfig(filename : string)
+ showConfig()
+ run()
- trainAuto(trainData, model)
≪create≫
featureExtractor
- bitsize : int
- segmentation type : string
- output location : string
- image location : string
- filenames : vector<string>
+ extract(filtersize : int, directory : string)
- filter(filtersize : int)
≪create≫
BSIFFilter
- size : int
- bits : int
- filter : double*
+ filtername : string
+ loadFilter(filtersize : int, bits : int)
+ generateHistogram(image : Mat) : vector<int>

Figure 3.1. TCL Detection C++ implementation class diagram

CHAPTER 4

RESULTS

4.1 Datasets

The datasets used for training, validation, and testing throughout the development of this method were taken from LiveDet-Iris 2017 [28]. Datasets in LiveDet-Iris 2017 are split into training and testing subsets. Additonally, the testing subset is further split into known and unknown subsets, indicating whether the presentation attack instruments used in the testing set were also present in the training set.

The dataset used for training the images was the Notre Dame Contact Lens Detection 2015 (NDCLD'15) dataset [8]. A portion (4800 images) of this dataset is included in LiveDet-Iris 2017, but to maintain consistency with the method proposed by Doyle and Bowyer, 7300 images from this dataset were used. Images were acquired with two different sensors – the IrisAccess LG4000 and IrisGuard AD100 – that are equally represented in the dataset. Each image has an ISO-compliant resolution of 640×480. The dataset includes images with no contact lenses, clear contact lenses, and textured contact lenses from five different brands: J&J, Ciba, Cooper, UCL, and ClearLab. The presence of clear contact lenses is a key difference from the LiveDet-Iris 2017 version of the dataset, in which only images without lenses or with textured lenses are included. The focus of this paper will be on differentiating between the clear/no lens case and the textured lens case.

In order to ensure that the iris PAD baseline was not limited in its ability to achieve a reasonable accuracy on other datasets, two additional datasets used in

TABLE 4.1

DATASET COMPOSITION

Dataset	Live Images	Textured Images	Lens Brands
NDCLD'15	4800	2500	5
Clarkson	2469	1122	15
IIITD	2250	1000	–

the LivDet-Iris 2017 competition, Clarkson and IIITD, were used for the validation and testing of models trained on NDCLD'15. The Clarkson dataset was collected using the LG IrisAccess EOU2200 and consists of live iris images, textured contact lens images, and printouts of iris images. Given that the focus of this paper is the detection of textured contact lenses, the printout images were not used. The training set, used in this paper, consisted of 2469 live images and 1122 textured lens images with fifteen different contact lens types. The IIITD dataset is the training portion of the combined IIITD-WVU dataset from LiveDet. It consists of 2250 real and 1000 textured contact lens images from the IIITD Contact Lens Iris database.

4.2 Hyperparameter Optimization

Hyperparameter optimization was performed using the `trainAuto()` function in OpenCV for the SVM models and a separate but similar implementation of an automatic training function for the multilayer perceptron and random forest models. The difference between optimized models and default models in accuracy on the training set can be seen in Fig. 4.1. The `trainAuto()` function in OpenCV uses k-fold cross-validation with 10 folds to select the hyperparameters for the SVM. These parameters include γ and c, which control the RBF shape and the penalty for outliers,

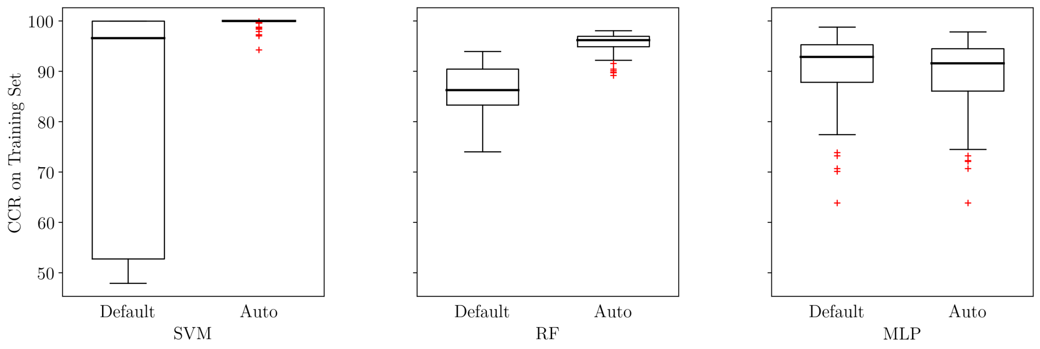


Figure 4.1. Performance on the training set when the default training function or automatic training function was used

respectively.

For the multilayer perceptron and random forest models, there was no automatic hyperparameter selection function implemented in OpenCV. Therefore, the function used to train the SVM was replicated within the `TCLManager` class. The function takes the training samples and randomly shuffles them prior to dividing them into ten different sets. Cross-validation is then used to select the best set of parameters for each model. For the multilayer perceptron, the only parameter that is adjusted is the size of the hidden layer. The size of the hidden layer was expressed as a function of the number of training samples, n, such that the size of the hidden layer was $n \times m$, where $m = \{1, 2, 4, 8\}$. For the random forest models, the depth of the tree and the minimum number of samples required to split a leaf node were adjusted. Six different values were tested for the depth d of the tree, $d = \{1, 5, 10, 15, 20, 15\}$. The minimum sample count was expressed as a percentage of the number of training samples, n, such that the minimum sample count was $p\%$ of n, where $p = \{1, 1.5, 2, 2.5\}$.

4.3 Base Case

Following Doyle *et al.*, a base case was tested with SVM models trained on BSIF with $n = 8$ and all available scales, giving 16 total models. SVM was selected as it was the highest performing model in the original paper. These models were trained on an 80:20 split of the NDCLD'15 dataset – 5840 images in the training set and 1460 images in the validation set. The models were ranked by their performance on the validation set and then added one at a time to an ensemble, as can be seen in Fig. 4.2. The highest validation performance was achieved with a 10 model ensemble, which gave a CCR of 99.86%.

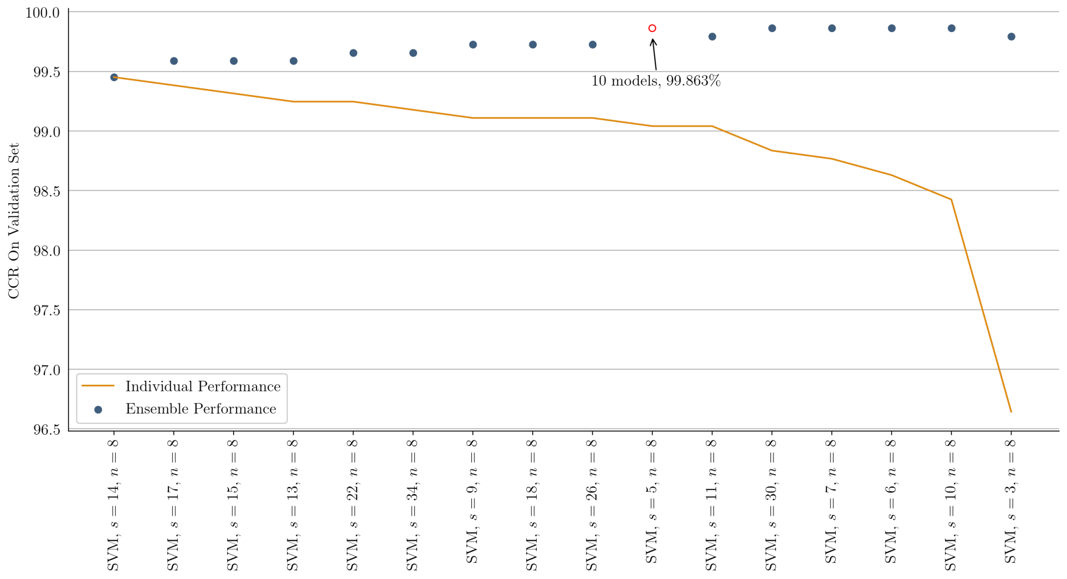


Figure 4.2: Building a classification ensemble on **Notre Dame** dataset.

This ensemble, trained on the NDCLD'15 dataset, was then tested in a **cross-dataset** scenario to assess how the proposed benchmark generalizes to unknown data.

The Clarkson and IIITD training partitions from the LivDet-Iris 2017 benchmark were used for that purpose. To estimate the variance of the test results, ten testing iterations were performed, with each iteration consisting of a randomly selected set of images that was half the size of the overall test set. As can be seen in Fig. 4.3, the ensemble trained on the NDCLD'15 dataset does not produce results that are on par with the LiveDet-Iris 2017 winner, which suggests that the generalization capabilities of a solution achieving an excellent CCR = 99.8% in the same-dataset scenario is limited. Therefore, additional BSIF filters and additional classifiers were considered in the extended case to improve the poor cross-dataset performance of the base case.

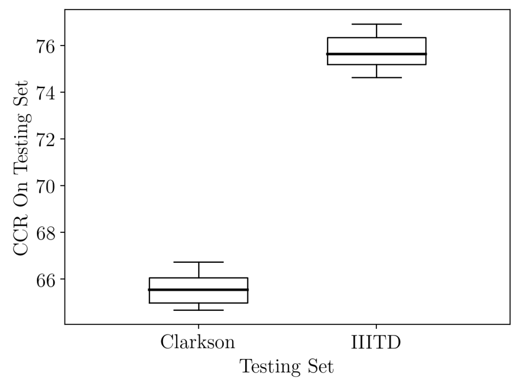


Figure 4.3. Box plots presenting the correct classification rates seen when validation is performed on one dataset (ND) and testing on others. Bold bars denote median values, height of each boxes equals to an inter-quartile range (IQR) spanning from the first (Q1) to the third (Q3) quartile, and the whiskers span from Q1-1.5*IQR to Q3+1.5*IQR.

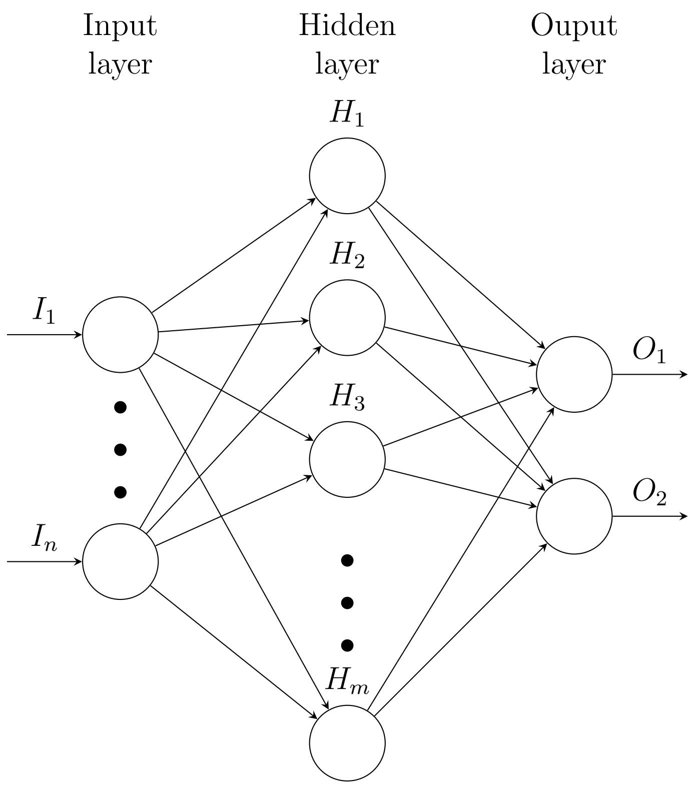


Figure 4.4. Multilayer perceptron structure

4.4 Extended Case

4.4.1 Model Training

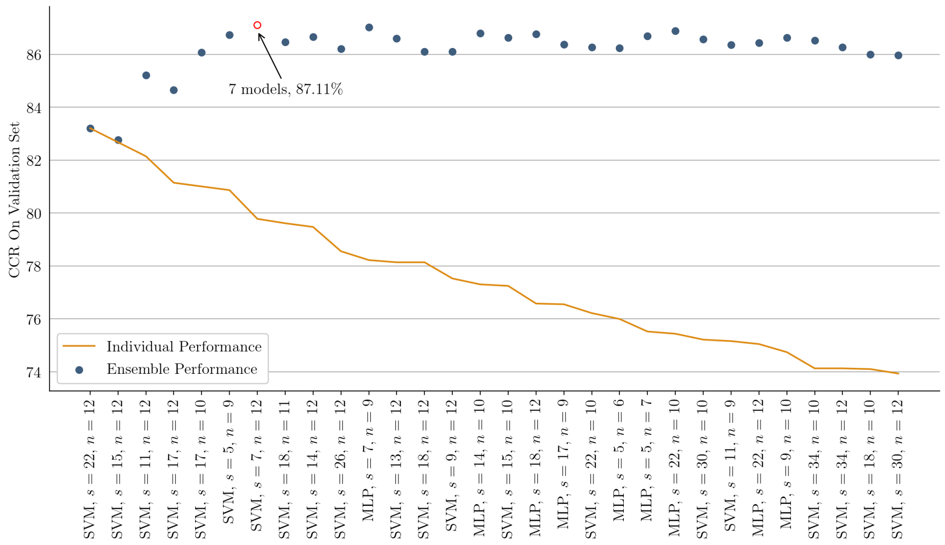
The NDCLD15 database was used to train an ensemble consisting of SVM, multilayer perceptron, and random forest models. The SVMs used C-Support Vector Classification with the radial basis function (RBF) as the kernel. The multilayer perceptron models consisted of an input layer with the number of neurons corresponding to the number of input features, a single hidden layer with the number of neurons equal to an integer multiple of the input layer, and an output layer consisting of two neurons, as shown in Fig. 4.4. The hyperparameter optimization performed for each model type – SVM, multilayer perceptron, and random forest – is described in 4.2. Further descriptions of each model type can be found in Appendix A.

A final ensemble of models, made available with this paper, was trained on the entire NDCLD'15 dataset. BSIF includes eight standard kernel scales and eight kernel depths. For $s = 3$, only the first four depth options are available, giving a total of 60 different scale and depth combinations. When combined with down-sampling, this gives 120 different feature sets for each image. For each combination of $s \in \{3, 5, 6, 7, 9, 10, 11, 13, 14, 15, 17, 18, 22, 26, 30, 34\}$ and $n \in \{5, 6, 7, 8, 9, 10, 11, 12\}$, three models (SVM, RF, and MLP) were trained on features extracted from the NDCLD'15 dataset, giving 360 total models available in the ensemble.

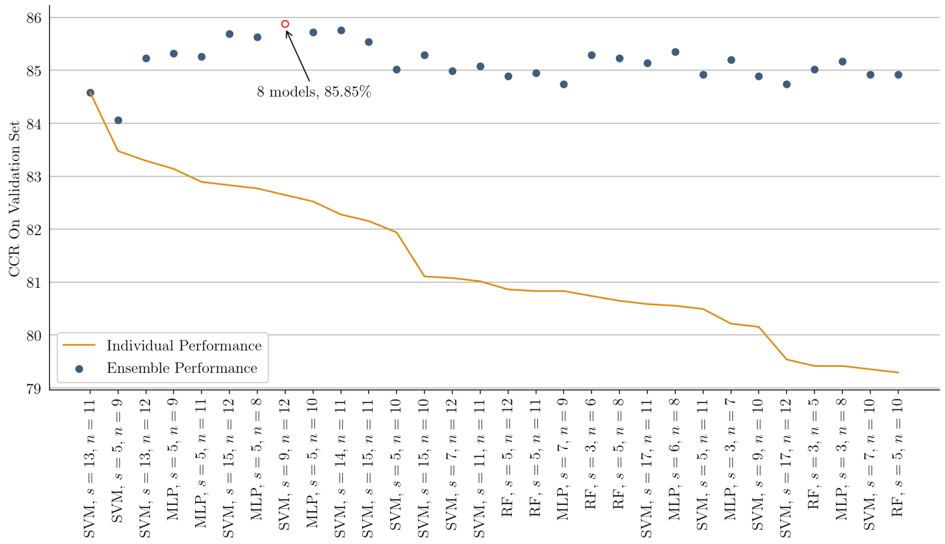
4.4.2 Validation Procedure

To select the strongest models, either the Clarkson or IIITD dataset was used as the validation set. The other dataset was excluded from the validation procedure to be used as the testing set. Each model was tested on the validation set and then ranked by the correct classification rate measured on the validation set. The models were then added one by one from strongest to weakest to a majority voting test on the validation set to determine the optimal number of models to use in the ensemble. For example, if a peak was found when using the top eight models for majority voting on the validation set, these top eight models were selected to test for ensemble performance on the testing set. This procedure ensures that the other dataset will be **unknown** to the algorithm and the results will be a true indicator of **cross-dataset** performance.

The results from validation on Clarkson and IIITD can be seen in Fig. 4.5 where both the individual model performance and ensemble performance are shown. For Clarkson, the highest performance (determined by the correct classification rate) comes when seven models are used in majority voting, giving a CCR of 87.11%. For IIITD, the highest performance comes when eight models are used in majority voting, giving a CCR of 85.88%. The individual performance of each of the selected models,



(a) Building a classification ensemble on **Clarkson** dataset.



(b) Building a classification ensemble on **IIITD** dataset.

Figure 4.5. Validation results for default BSIF

TABLE 4.2

MODELS SELECTED FROM VALIDATION ON CLARKSON

Model	CCR	APCER	BPCER
SVM, $s = 22$, $n = 12$	83.21	33.96	8.99
SVM, $s = 15$, $n = 12$	82.68	22.99	14.74
SVM, $s = 11$, $n = 12$	82.15	17.91	17.82
SVM, $s = 17$, $n = 12$	81.15	16.76	19.81
SVM, $s = 17$, $n = 10$	81.01	21.30	17.94
SVM, $s = 5$, $n = 9$	80.87	28.16	15.03
SVM, $s = 7$, $n = 12$	79.78	18.18	21.14

including their APCER and BPCER, can be viewed in tables 4.2 and 4.3.

4.4.3 Cross-Dataset Testing

After the ensemble of models was selected on the validation set, it was used to classify the other dataset, which served as the testing set. To estimate the variance of the testing results, ten testing iterations were performed, with each iteration consisting of a randomly selected set of images that was half the size of the overall test set. The results of this test can be seen in Fig. 4.6. The 7 models selected using Clarkson as the validation set were able to achieve a median correct classification rate of 84.45% on IIITD and the 8 models selected using IIITD were able to achieve a median correct classification rate of 84.11% on Clarkson.

These results are close to the performance achieved by the LivDet-Iris 2017 winner (90% on Clarkson dataset and 83% on IIITD dataset). Such comparison should be

TABLE 4.3

MODELS SELECTED FROM VALIDATION ON IIITD

Model	CCR	APCER	BPCER
SVM, $s = 13$, $n = 11$	84.58	37.00	5.82
SVM, $s = 5$, $n = 9$	83.48	50.40	1.47
SVM, $s = 13$, $n = 12$	83.29	40.10	6.31
MLP, $s = 5$, $n = 9$	83.14	44.90	1.42
MLP, $s = 5$, $n = 11$	82.89	44.40	1.69
SVM, $s = 15$, $n = 12$	82.83	50.00	2.58
MLP, $s = 5$, $n = 8$	82.77	45.00	2.44
SVM, $s = 9$, $n = 12$	82.65	39.90	7.33

done with care, as the LivDet-Iris 2017 paper reports combined results of textured contact lens **and** printouts detection. However, if iris printouts are – on average – easier to detect than textured contact lenses, as suggested by the LivDet-Iris 2017 results, the obtained accuracy for this open source benchmark compares favorably to the LivDet winner. These results show that a larger ensemble can be more robust to cross-dataset tests.

4.4.4 Domain Specific BSIF

The extended case procedure was repeated for domain specific BSIF, which has been shown to out-perform default BSIF for iris recognition [6]. The domain specific BSIF filters were trained from iris recognition image patches and included twelve kernel sizes and eight different kernel depths, giving a total of 192 feature sets per image, including down-sampling. As described in 4.4.1, three models (SVM, RF, and MLP)

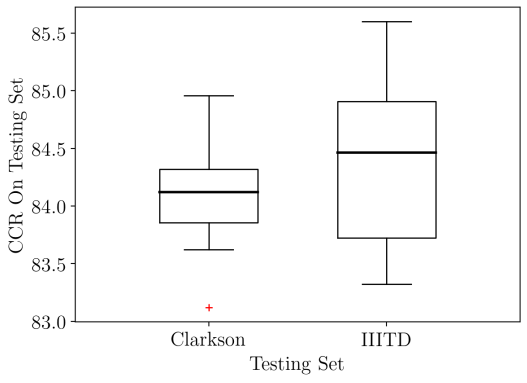


Figure 4.6. Box plots presenting the correct classification rates seen when validation is performed on one dataset and testing on the other. Bold bars denote median values, height of each boxes equals to an inter-quartile range (IQR) spanning from the first (Q1) to the third (Q3) quartile, and the whiskers span from Q1-1.5*IQR to Q3+1.5*IQR.

were trained on features extracted from the NDCLD'15 dataset for each combination of $s \in \{5, 7, 9, 10, 11, 13, 14, 15, 17, 18, 19, 21, 22, 26, 27, 30, 33, 34, 38, 39, 42, 54, 66, 78\}$ and $n \in \{5, 6, 7, 8, 9, 10, 11, 12\}$, giving 576 total models available in the ensemble.

As in 4.4.2, either the Clarkson or IIITD dataset was used as the validation set to select the strongest models from the ensemble. The other dataset was excluded from this procedure and later used as the testing set. The results of validation on Clarkson and IIITD for the domain-specific models can be seen in Fig. 4.7. For Clarkson, the highest performance comes when twelve models are used in majority voting, giving a CCR of 84.15%. For IIITD, the highest performance comes when three models are used in majority voting, giving a CCR of 83.29%. The individual performance of each of the selected models, including their APCER and BPCER, can be viewed in tables 4.4 and 4.5.

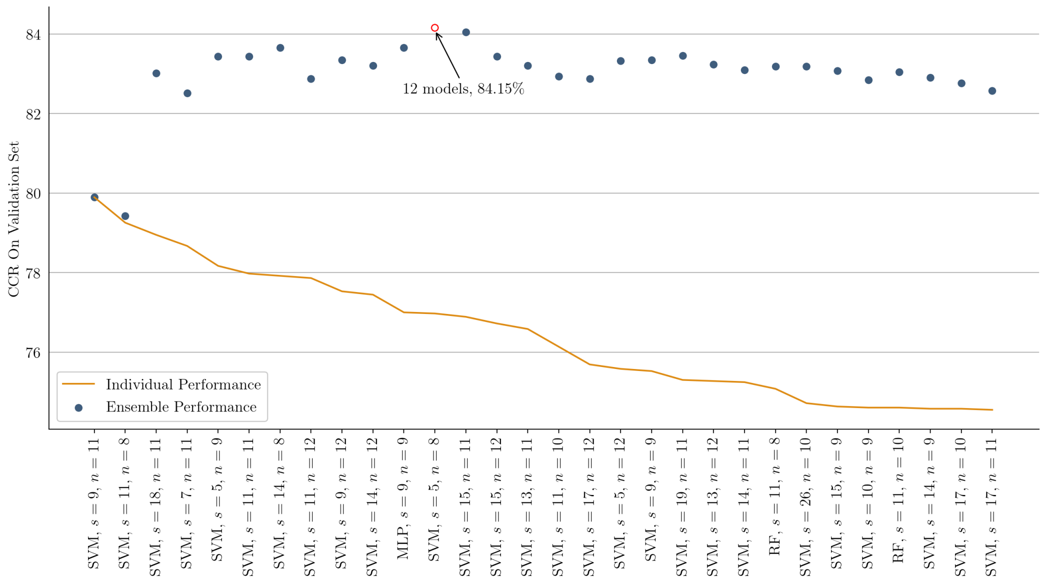
These selected ensembles were used to classify the other dataset, as done in 4.4.3 for default BSIF. To estimate the variance of the testing results, ten testing iterations were performed as before, with each iteration consisting of a randomly selected set

TABLE 4.4

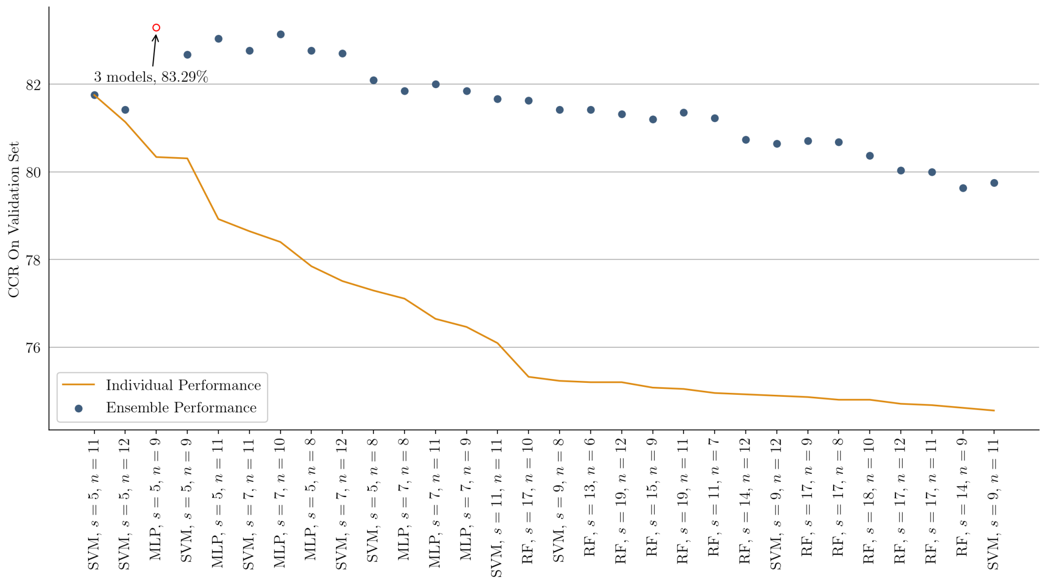
MODELS SELECTED FROM VALIDATION ON CLARKSON WITH DOMAIN-SPECIFIC BSIF

Model	CCR	APCER	BPCER
SVM, $s = 9$, $n = 11$	79.89	49.55	6.72
SVM, $s = 11$, $n = 8$	79.25	31.28	15.96
SVM, $s = 18$, $n = 11$	78.95	40.73	12.11
SVM, $s = 7$, $n = 11$	78.67	38.41	13.57
SVM, $s = 5$, $n = 9$	78.17	31.10	17.62
SVM, $s = 11$, $n = 11$	77.97	65.78	2.15
SVM, $s = 14$, $n = 8$	77.92	43.49	12.35
SVM, $s = 11$, $n = 12$	77.86	61.14	4.41
SVM, $s = 9$, $n = 12$	77.53	26.20	20.78
SVM, $s = 14$, $n = 12$	77.44	39.39	14.90
SVM, $s = 9$, $n = 9$	77.00	48.57	11.38
MLP, $s = 5$, $n = 8$	76.97	19.96	24.42

of images that was half the size of the overall test set. The results of this test can be seen in Fig. 4.8. The 12 models selected using Clarkson were able to achieve a median correct classification rate of 79.07% on IIITD and the 3 models selected using IIITD were able to achieve a median correct classification rate of 76.80% on Clarkson. Therefore, domain-specific BSIF did not improve the results for this **cross-dataset** case. Domain-specific BSIF was trained for iris recognition, so it is possible that this is why the improvements seen in [6] were not seen in the PAD case.



(a) Building a classification ensemble on **Clarkson** dataset.



(b) Building a classification ensemble on **IIITD** dataset.

Figure 4.7. Validation results for domain-specific BSIF

TABLE 4.5

MODELS SELECTED FROM VALIDATION ON IIITD

WITH DOMAIN-SPECIFIC BSIF

Model	CCR	APCER	BPCER
SVM, $s = 5$, $n = 11$	81.75	56.40	1.29
SVM, $s = 5$, $n = 12$	81.14	49.90	5.07
MLP, $s = 5$, $n = 9$	80.34	54.40	4.22

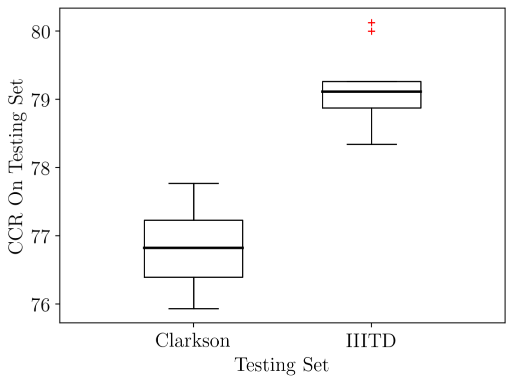


Figure 4.8. Box plots presenting the correct classification rates seen when validation is performed on one dataset and testing on the other. Bold bars denote median values, height of each boxes equals to an inter-quartile range (IQR) spanning from the first (Q1) to the third (Q3) quartile, and the whiskers span from Q1-1.5*IQR to Q3+1.5*IQR.

CHAPTER 5

DEMONSTRATION

A demonstration was prepared to showcase the capabilities of the baseline iris PAD solution when combined with a low cost USB imaging device. The program used for the demonstration uses the functions implemented for the open source baseline and shows the ease with which one may extend and adapt the baseline for different purposes.

5.1 Camera and SDK

The camera selected for the demonstration was the IriShield^{TM}-USB 2120U[1] due to its portability and relatively low cost. Images are taken in near infrared (NIR) and comply with ISO 19794-6 as they are 640×480 8-bit grayscale images. Illumination is provided by a NIR LED at the top of the device.

With the camera, Iritech, Inc. provides the IDDK 2000 API written in C++. This API provides the functions required to configure and control all functionalities of the device, such as the initialization of the camera and the control of the capturing process. The API stores images in an `unsigned char` array format, so the only change required to enable compatibility with the iris PAD baseline was to convert this format into the `Mat` format of OpenCV.

[1]`https://www.iritech.com/products/hardware/irishield-series`

5.2 Capabilities

The demonstration is able to acquire and classify an iris image. To acquire an image, the IDDK 2000 API is used. The acquisition procedure is able to recognize when there is an iris (or printout) in front of the camera. Once an iris is recognized, the camera will take a sequence of images, with the best one selected for further use. This image is shown on screen to indicate what will be used for classification.

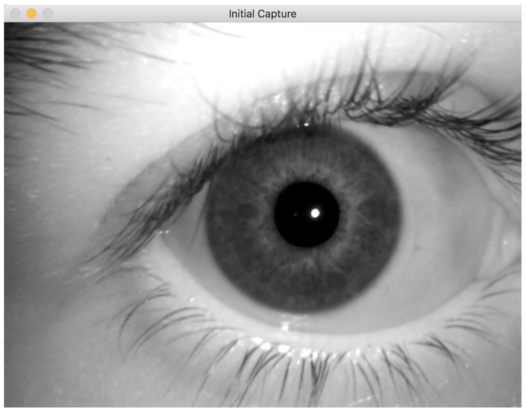
An ensemble of models is then used to classify the image as a real iris or a spoof. Two different ensembles are used for the demonstration: for printouts, an ensemble was selected on the Warsaw dataset and for textured contact lenses, an ensemble was selected on the Clarkson dataset. After classification, the image is modified with text showing the result of the majority voting as well as the votes for each class. This image is displayed to the user to indicate the overall result of the test.

5.3 Model Selection

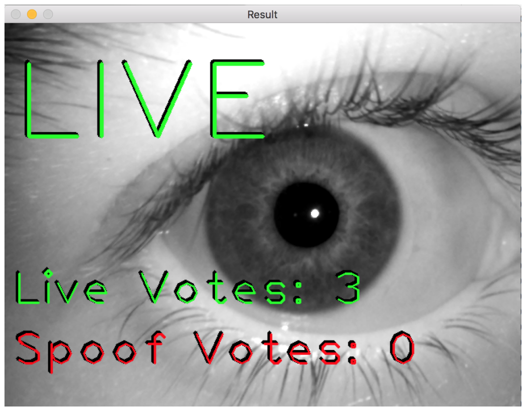
The model selection for the textured contact lens version of the demonstration was done on the Clarkson dataset. Results for this can be seen in 4.5.

To select models for use in the printout version of the demonstration, the validation procedure described in Chapter 4 was performed on the Warsaw dataset, which was included in LiveDet-Iris 2017 and consists of images of live irises in addition to images of printouts. For validation, only the training portion of the set is used, which consists of 1844 images of irises and 2669 images of printouts, all acquired using the IrisGuard AD100. The results of validation on Warsaw can be seen in Fig. 5.2: a performance of 95.76% was achieved with an ensemble of three models.

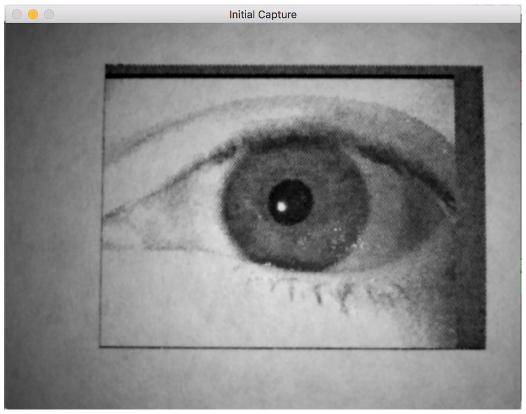
As can be seen in table 5.1, the models that perform well on printouts tend to have smaller BSIF scales, s, than models that performed well on textured lenses (seen in tables 4.2 and 4.3). This is possibly due to the high frequency content of a



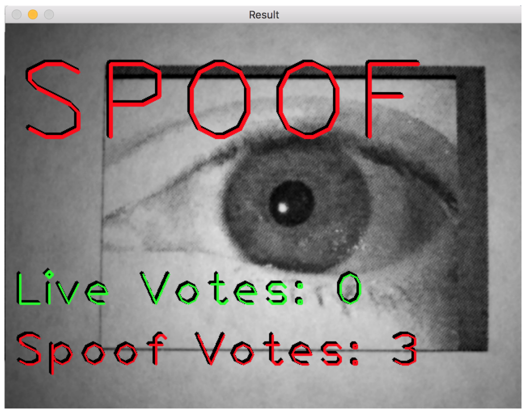
(a) Live iris initial capture



(b) Live iris result



(c) Printout initial capture



(d) Printout result

Figure 5.1. Demonstration operation with a live iris and a printout

printout, which would be better identified by a smaller filter scale. The difference in the best performing models across different PAI is confirmed by testing the models selected on Warsaw on the other two validation sets. As can be seen in Fig. 5.3 , the cross-PAI performance is far below that of the same-PAI performance seen in Chapter 4; therefore, two versions of the demonstration were prepared.

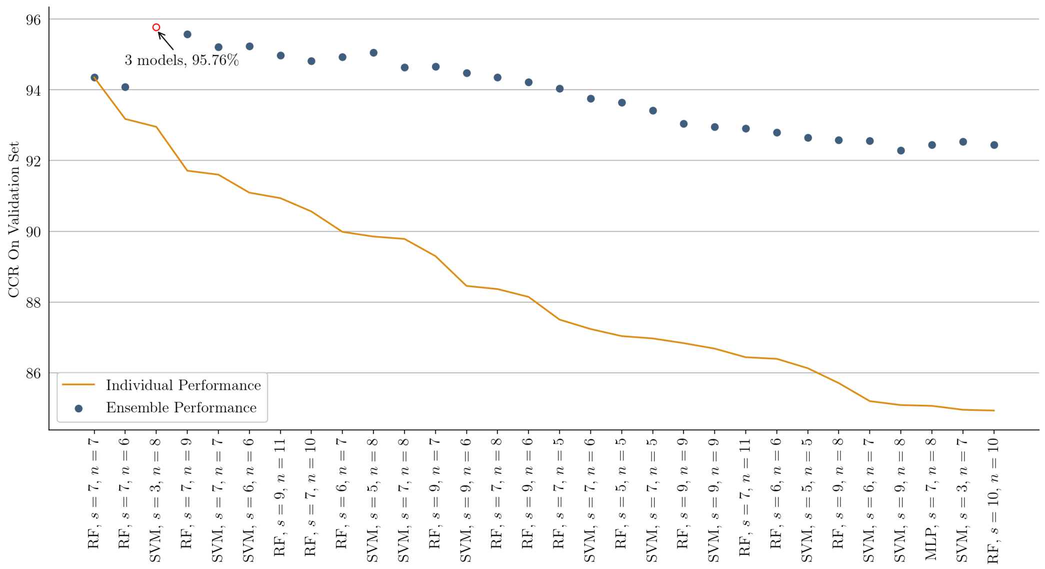


Figure 5.2: Building a classification ensemble on **Warsaw** dataset.

5.4 Demonstration Conclusions

The simple demonstration shown here illustrates the benefits of having an open source baseline for iris PAD: it can be quickly incorporated into different systems and easily modified to accomodate other PAI (printouts in this case).

TABLE 5.1

MODELS SELECTED FROM VALIDATION ON WARSAW

Model	CCR	APCER	BPCER
RF, $s = 7$, $n = 7$	94.35	7.38	3.15
RF, $s = 7$, $n = 6$	93.18	9.18	3.42
SVM, $s = 3$, $n = 8$	92.95	5.25	9.65

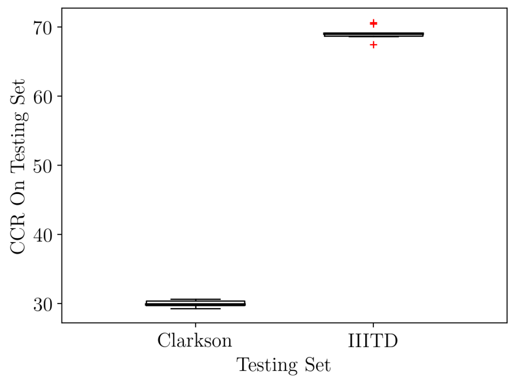


Figure 5.3. Box plots presenting the correct classification rates seen when validation is performed on printouts (Warsaw) and testing on textured contact lenses (Clarkson and IIITD). Bold bars denote median values, height of each boxes equals to an inter-quartile range (IQR) spanning from the first (Q1) to the third (Q3) quartile, and the whiskers span from Q1-1.5*IQR to Q3+1.5*IQR.

CHAPTER 6

CONCLUSIONS

6.1 Open Source Baseline

This thesis offers the first, known to us, open-source software solution for iris presentation attack detection. It is based on a recent and effective methodology of using Binary Statistical Image Features and an ensemble of classifiers to detect textured contact lenses.

This software allows for retraining the ensemble with other datasets, defining which classifiers form the ensemble, and calculating BSIF-based features that can be used to test other classifiers worth adding to the ensemble. The long-term goal of this effort is to build an open-source baseline methodology for iris PAD, for instance for the next editions of the LivDet-Iris competitions, starting from a recent and effective algorithm of textured contact lens detection.

6.2 Novel Dataset Performance

A trained ensemble of classifiers, added to this initial version, achieves a correct classification rate of 84% on challenging cross-dataset tests, in a close-set scenario and with only best guess iris segmentation required. This result is similar to the cross-dataset classification accuracy achieved by the most recent LivDet-Iris competition winner, which makes this implementation a useful benchmark for iris PAD.

APPENDIX A

MODEL DESCRIPTIONS

Here the models chosen for the baseline are described in more detail.

A.1 Support Vector Machine

A support vector machine (SVM) is a binary classifier formed by a separating hyperplane[1].

$$f(x) = \beta_0 + \beta^T x \tag{A.1}$$

This hyperplane will be optimal for SVM, meaning that the margin between the separator and any examples is at a maximum. The optimal hyperplane is typically represented such that $f(x)$ gives 1 for all support vectors.

$$|\beta_0 + \beta^T x| = 1 \tag{A.2}$$

The support vectors are the training examples that are closest to the hyperplane.

If the training data is not linearly separable, it can be transformed using a kernel. In this paper, the radial basis function is used as the kernel.

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2} \tag{A.3}$$

The kernel places the data in a new space where it may be linearly separable. For example, a linear discriminator can serve as a circular discriminator if the transfor-

[1]`https://docs.opencv.org/3.4/d1/d73/tutorial_introduction_to_svm.html`

mation $x_1^2 + x_2^2$ is used. One important feature of kernel transformations for SVM is the kernel trick. Training a SVM involves taking the inner product between pairs of data in the feature space. This inner product can be computationally cheaper to compute than the computation of the individual kernel transformations; thus, the feature space is only implicitly present because the kernel trick allows for faster computation without explicit calculation of the transformation.

A.2 Random Forest

The random forest classifier is built from many different decision trees[2]. A decision tree performs a series of tests on an input sample to reach an output classification. Each node within the tree – a decision node – looks at a portion m of the M input variables associated with the sample. The best split on these m variables, i.e. the test that best separates the different classes, is used to split the node. The tree is grown until no more separations can be made.

To create the decision trees for a random forest, a training set is created for each tree through sampling with replacement (bootstrapping). For each training set, a third of the samples are excluded and kept as out-of-bag (oob) data, which can then be used to provide an estimation of the error as the forest is trained. After all trees have been constructed, the forest can be used to classify new inputs. The classification given to an input is the classification that receives the most votes when classified by each tree individually.

[2]`https://www.stat.berkeley.edu/users/breiman/RandomForests/cc_home.htm`

A.3 Multilayer Perceptron

A multilayer perceptron is a type of feed-forward artificial neural network that has a number of hidden layers in addition to the input and output layers[3]. The neurons in a multilayer perceptron are interconnected, with each having inputs from the previous layer and outputs to the next layer. Each neuron takes a weighted sum of the inputs from the previous layer and transforms it using an activation function. The activation function for the hidden layer neurons in this paper was a symmetrical sigmoid function,

$$f(x) = \beta * \frac{1 - e^{-\alpha x}}{1 + e^{\alpha x}} \tag{A.4}$$

with α and β set to one.

To train a neural network, backpropagation – in which the gradient of the error is calculated with respect to the weights in the network – is used. The general gradient descent approach can then be taken to update the weights in the network. Many different approaches to the structure of a neural network exist, but generally the input layer consists of the same number of neurons as features. A multilayer perceptron does not output a binary classification; instead, two output neurons must be used with one representing the positive class and the other representing the negative class.

[3]`https://docs.opencv.org/2.4/modules/ml/doc/neural_networks.html`

BIBLIOGRAPHY

1. G. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly, 2008.

2. C. Chen and A. Ross. A multi-task convolutional neural network for joint iris detection and presentation attack detection. In *IEEE Winter Conf. on Applications of Computer Vision (WACV)*, pages 44–51, March 2018. doi: 10.1109/WACVW.2018.00011.

3. A. Czajka. Pupil dynamics for iris liveness detection. *IEEE Trans. Inf. Forens. Security*, 10(4):726–735, April 2015. ISSN 1556-6013. doi: 10.1109/TIFS.2015.2398815.

4. A. Czajka and K. W. Bowyer. Presentation attack detection for iris recognition: An assessment of the state-of-the-art. *ACM Computing Surveys*, 51(4):86:1–86:35, July 2018. ISSN 0360-0300. doi: 10.1145/3232849. URL `http://doi.acm.org/10.1145/3232849`.

5. A. Czajka, Z. Fang, and K. Bowyer. Iris presentation attack detection based on photometric stereo features. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 877–885, Jan 2019. doi: 10.1109/WACV.2019.00098.

6. A. Czajka, D. Moreira, K. Bowyer, and P. Flynn. Domain-specific human-inspired binarized statistical image features for iris recognition. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Jan 2019.

7. J. Daugman. How iris recognition works. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1):21–30, 2004.

8. J. S. Doyle and K. W. Bowyer. Robust detection of textured contact lenses in iris recognition using bsif. *IEEE Access*, 3:1672–1683, 2015. ISSN 2169-3536. doi: 10.1109/ACCESS.2015.2477470.

9. J. S. Doyle, K. W. Bowyer, and P. J. Flynn. Variation in accuracy of textured contact lens detection based on sensor and lens pattern. In *IEEE Int. Conf. on Biometrics: Theory Applications and Systems (BTAS)*, pages 1–7, Arlington, VA, USA, September 2013. IEEE. doi: 10.1109/BTAS.2013.6712745.

10. J. S. Doyle, P. J. Flynn, and K. W. Bowyer. Automated classification of contact lens type in iris images. In *2013 Int. Conf. on Biometrics (ICB)*, pages 1–6, Madrid, Spain, June 2013. IEEE. doi: 10.1109/ICB.2013.6612954.

11. J. Galbally, J. Ortiz-Lopez, J. Fierrez, and J. Ortega-Garcia. Iris liveness detection based on quality related features. In *2012 5th IAPR Int. Conf. on Biometrics (ICB)*, pages 271–276, New Delhi, India, March 2012. IEEE. doi: 10.1109/ICB.2012.6199819.

12. J. Kannala and E. Rahtu. Bsif: Binarized statistical image features. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1363–1366, Nov 2012.

13. O. V. Komogortsev, A. Karpov, and C. D. Holland. Attack of mechanical replicas: Liveness detection with eye movements. *IEEE Trans. Inf. Forens. Security*, 10 (4):716–725, April 2015. ISSN 1556-6013. doi: 10.1109/TIFS.2015.2405345.

14. J. Komulainen, A. Hadid, and M. Pietikainen. Generalized textured contact lens detection by extracting BSIF description from Cartesian iris images. In *IEEE Intl. Joint Conference on Biometrics*, 2014.

15. E. C. Lee and K. R. Park. Fake iris detection based on 3d structure of iris pattern. *Int. Journal of Imaging Systems and Technology*, 20(2):162–166, 2010. ISSN 1098-1098. doi: 10.1002/ima.20227. URL `http://dx.doi.org/10.1002/ima.20227`.

16. D. Menotti, G. Chiachia, A. Pinto, W. Schwartz, H. Pedrini, A. Falcao, and A. Rocha. Deep representations for iris, face, and fingerprint spoofing detection. *IEEE Trans. Inf. Forens. Security*, 10(4):864–879, April 2015.

17. N. Othman, B. Dorizzi, and S. Garcia-Salicetti. Osiris: An open source iris recognition software. *Pattern Recognition Letters*, 2015.

18. J. H. Park and M. G. Kang. Multispectral iris authentication system against counterfeit attack using gradient-based image fusion. *Optical Engineering*, 46 (11):117003–117003–14, 2007. doi: 10.1117/1.2802367. URL `http://dx.doi.org/10.1117/1.2802367`.

19. R. Raghavendra and C. Busch. Presentation attack detection on visible spectrum iris recognition by exploring inherent characteristics of light field camera. In *IEEE Int. Joint Conf. on Biometrics (IJCB)*, pages 1–8, Clearwater, FL, USA, Sept 2014. IEEE. doi: 10.1109/BTAS.2014.6996226.

20. K. B. Raja, R. Raghavendra, and C. Busch. Video presentation attack detection in visible spectrum iris recognition using magnified phase information. *IEEE Trans. Inf. Forens. Security*, 10(10):2048–2056, October 2015. ISSN 1556-6013. doi: 10.1109/TIFS.2015.2440188.

21. L. Thalheim, J. Krissler, and P.-M. Ziegler. Biometric Access Protection Devices and their Programs Put to the Test, Available online in c't Magazine, No. 11/2002, p. 114. on-line, 2002.

22. S. Thavalengal, T. Nedelcu, P. Bigioi, and P. Corcoran. Iris liveness detection for next generation smartphones. *IEEE Trans. Cons. Elect.*, 62(2):95–102, May 2016. ISSN 0098-3063. doi: 10.1109/TCE.2016.7514667.

23. M. Trokielewicz, A. Czajka, and P. Maciejewicz. Human iris recognition in post-mortem subjects: Study and database. In *IEEE Int. Conf. on Biometrics: Theory Applications and Systems (BTAS)*, pages 1–6, Niagara Falls, NY, USA, Sept 2016. IEEE. doi: 10.1109/BTAS.2016.7791175.

24. F. M. Villalbos-Castaldi and E. Suaste-Gómez. In the use of the spontaneous pupillary oscillations as a new biometric trait. In *Int. Workshop on Biometrics and Forensics*, pages 1–6, Valletta, Malta, March 2014. IEEE. doi: 10.1109/IWBF.2014.6914259.

25. Z. Wei, T. Tan, and Z. Sun. Synthesis of large realistic iris databases using patch-based sampling. In *Int. Conf. on Pattern Recognition (ICPR)*, pages 1–4, Tampa, FL, USA, Dec 2008. IEEE. doi: 10.1109/ICPR.2008.4761674.

26. D. Yadav, J. Doyle, and M. Vatsa. Unraveling the effect of textured contact lenses on iris recognition. *IEEE Transactions on Information Forensics and Security*, 9(5):851–862, 2014.

27. D. Yambay, J. S. Doyle, K. W. Bowyer, A. Czajka, and S. Schuckers. Livdet-iris 2013 - iris liveness detection competition 2013. In *IEEE Int. Joint Conf. on Biometrics (IJCB)*, pages 1–8, Clearwater, FL, USA, Sept 2014. IEEE. doi: 10.1109/BTAS.2014.6996283.

28. D. Yambay, B. Becker, N. Kohli, D. Yadav, A. Czajka, K. W. Bowyer, S. Schuckers, R. Singh, M. Vatsa, A. Noore, D. Gragnaniello, C. Sansone, L. Verdoliva, L. He, Y. Ru, H. Li, N. Liu, Z. Sun, and T. Tan. LivDet Iris 2017 – iris liveness detection competition 2017. In *IEEE Int. Joint Conf. on Biometrics (IJCB)*, pages 1–6, Denver, CO, USA, 2017. IEEE.

29. D. Yambay, B. Walczak, S. Schuckers, and A. Czajka. Livdet-iris 2015 - iris liveness detection competition 2015. In *IEEE Int. Conf. on Identity, Security and Behavior Analysis (ISBA)*, pages 1–6, New Delhi, India, Feb 2017. IEEE. doi: 10.1109/ISBA.2017.7947701.

This document was prepared & typeset with pdfLaTeX*, and formatted with* NDdiss2$_\varepsilon$ *classfile (v3.2017.2[2017/05/09])*