Open Source Biometric Recognition

Joshua C. Klontz
Michigan State University
East Lansing, MI, U.S.A
`klontzjo@msu.edu`

Brendan F. Klare
Noblis
Falls Church, VA, U.S.A
`brendan.klare@noblis.org`

Scott Klum
Michigan State University
East Lansing, MI, U.S.A
`klumscot@msu.edu`

Anil K. Jain
Michigan State University
East Lansing, MI, U.S.A
`jain@cse.msu.edu`

Mark J. Burge

Falls Church, VA, U.S.A.
`burge@ieee.org`

Abstract

The biometrics community enjoys an active research field that has produced algorithms for several modalities suitable for real-world applications. Despite these developments, there exist few open source implementations of complete algorithms that are maintained by the community or deployed outside a laboratory environment. In this paper we motivate the need for more community-driven open source software in the field of biometrics and present OpenBR *as a candidate to address this deficiency. We overview the OpenBR software architecture and consider still-image frontal face recognition as a case study to illustrate its strengths and capabilities. All of our work is available at* `www.openbiometrics.org`.

1. Introduction

Tools for collaborative software development have improved markedly over recent years thanks to sites like *GitHub* and *Launchpad* that provide free hosting services for open source projects. Despite the prevalence of these tools, we often observe a wide gap between methods published in the literature and implementations available in open source software. This increases the difficulty of reproducing research findings, a problem especially evident in computational fields like computer vision and machine learning [20], where a specific algorithm tailored to solve a particular dataset can constitute the primary technical achievement of a paper.

Despite the prevalence of tools for collaborative software development and the growing importance of biometrics to address a variety of security and identity problems, there currently does not exist a widely recognized open source framework tailored explicitly to the needs of the biometrics research community. In this work we present the Open Source Biometric Recognition (*OpenBR*) collaboratory that aspires to fill this gap. OpenBR provides tools to design and evaluate new biometric algorithms and an interface to incorporate biometric technology into end-user applications. While the software is designed with modality independence in mind, our development has focused on facial recognition research as a starting point to illustrate the importance and utility of this collaborative framework. As a result, this paper will focus on still-image face recognition as a case study to demonstrate existing capability and benefits to the research community.

1.1. Related Work

Table 1 reviews some of the most prominent open source face recognition software projects. Three criteria are applied that gauge if a project (i) implements modern algorithms, (ii) is under active development, and (iii) is readily deployable in applications. Two particularly notable solutions from Colorado State University (CSU) [17] and OpenCV [4] are discussed here further.

The CSU baseline algorithm suite includes two recently published algorithms, local region PCA (LRPCA) [17] and LDA with color spaces and cohort normalization (CohortLDA) [16]. The framework is written in Python and R with installation instructions provided for Mac and Windows. Scripts are included to run the algorithms against the GBU [17] and LFW [8] datasets, though programming language requirements and the lack of a well defined API make it difficult to incorporate their work into new applications. Source code is released as a zipped archive and it is unclear how developers should contribute back to the project.

The OpenCV library recently received a third-party source code contribution that adds the Eigenface [22], Fisherface [2], and LBP [1] face recognition algorithms. The

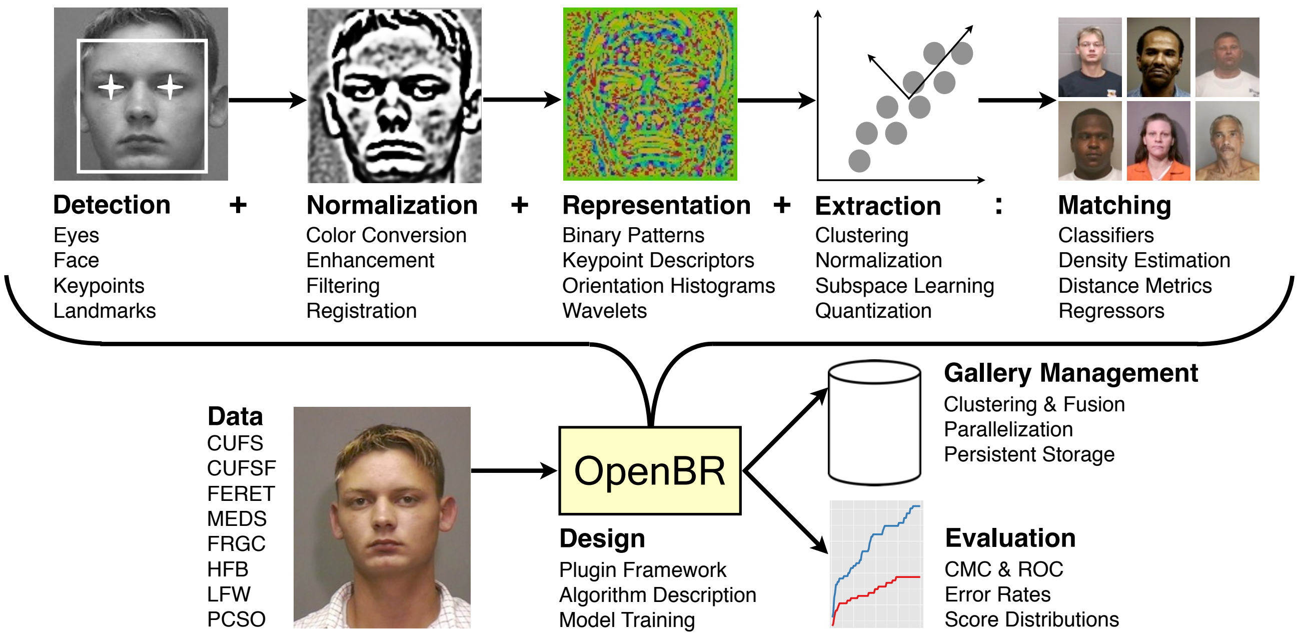

Figure 1: Overview of OpenBR capabilities.

Project	Modern	Active	Deployable
CSU [17]	Yes	No	No
OpenCV [4]	No	Yes	Yes
OpenBR	Yes	Yes	Yes

Table 1: Existing open source face recognition software. A project is considered *modern* if it incorporates peer-reviewed methods published in the last five years, *active* if it has source code changes made within the last six months, and *deployable* if it exposes a public API.

library is written in C++ with installation instructions provided for Windows, Linux/Mac, Android, and iOS. Documentation and examples demonstrate how to interact with the face recognition API, though it lacks modern face recognition methods and fine-grained eye localization. While the project enjoys daily source code improvements and an extensive developer network, it lacks of-the-shelf functionality useful to biometric researchers, including model training, gallery management, algorithm evaluation, cross-validation, and a concise syntax to express algorithms – all features available in OpenBR.

1.2. Overview

OpenBR is introduced in Section 2 as a project to facilitate open source biometrics research, and its architecture is detailed in Section 3. Section 4 describes the OpenBR face recognition capability, and its performance is evaluated on still-image frontal face datasets in Section 5. Novel contributions include a free open source framework for biometrics research, an embedded language for algorithm design, and a compact template representation and matching strategy that demonstrates competitive performance on academic datasets and the Face Recognition Vendor Test 2012.

2. The OpenBR Collaboratory

OpenBR is a framework for investigating new biometric modalities, improving existing algorithms, interfacing with commercial systems, measuring recognition performance, and deploying automated systems. The project is designed to facilitate rapid algorithm prototyping, and features a mature core framework, flexible plugin system, and support for open and closed source development. Figures 1 and 2 illustrate OpenBR functionality and software components.

OpenBR originated within The MITRE Corporation from a need to streamline the process of prototyping and evaluating algorithms. The project was later published as open source software under the *Apache 2* license and is free for academic and commercial use.

2.1. Software Requirements

OpenBR is written in a portable subset of *ISO C++* and is known to work with all popular modern C++ compilers including *Clang*, *GCC*, *ICC*, *MinGW-w64*, and *Visual Studio*. The project is actively maintained on Windows, Mac, and Linux, with ports to Android, iOS, and other platforms known to be within reach. OpenBR requires

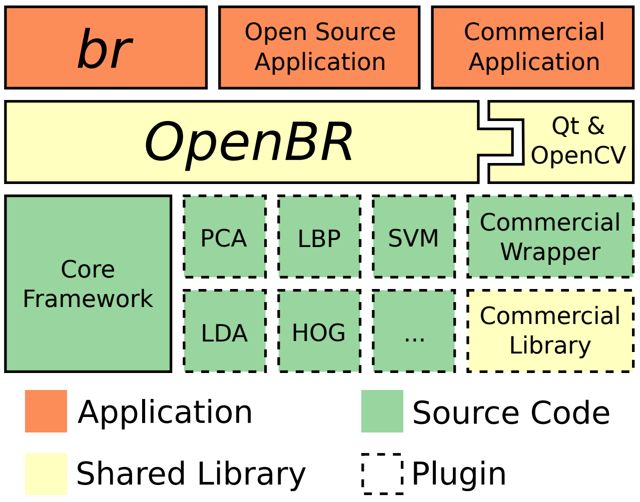

Figure 2: The OpenBR solution stack. The two principal software artifacts are the shared library `openbr` and command line application `br`.

the *OpenCV* computer vision library, *Qt* application framework, and *CMake* cross-platform build system. Complete build documentation is available online.

2.2. Language for Image Processing

Arguably the most important technical achievement of this project is a new language for image recognition used to express algorithms. Each word in the language corresponds to a plugin that performs a specific transformation on an image, allowing for highly decoupled development of the individual steps in an algorithm. These words are combined into an *algorithm description* that unambiguously defines template enrollment and comparison. The language is flexible enough to readily express most existing open source face recognition algorithms without additional development effort.

Embedding this language within the project realizes the benefits of compilation-free algorithm design and parameter exploration while simultaneously enjoying the efficiency and deployability of native software. In practice, we have also found that the language offers a concise, yet explicit, way of describing an algorithm to other researchers familiar with the project. The language will be introduced in further detail by example over the remainder of this paper.

2.3. Command Line Interface

The primary means of leveraging OpenBR functionality is the `br` command line application, an isomorphic wrapper to the `openbr` API. Entering a `br` command should be thought of as writing a short program, arguments can be specified in any quantity and are executed in the order they are given.

```
$ br -algorithm 'Open+Cvt(Gray)+Cascade
    (FrontalFace)+ASEFEyes+Affine
    (128,128,0.33,0.45)+CvtFloat+PCA
    (0.95):Dist(L2)' -train BioID/img
    Eigenfaces
```

Figure 3: Training Eigenfaces on the BioID dataset [10]. The algorithm description is wrapped in quotes to avoid unintended interpretation by the shell preprocessor.

```
$ br -algorithm Eigenfaces -path MEDS/
    img -compare MEDS/sigset/
    MEDS_frontal_target.xml MEDS/sigset/
    MEDS_frontal_query.xml scores.mtx
```

Figure 4: Computing a similarity matrix on the MEDS dataset [6] using the algorithm trained in Figure 3.

The capability of `br` is best illustrated through examples. In the interest of brevity, we omit many supported operations including clustering, cross-validation, gallery management, and score level fusion, though details of all OpenBR operations can be found online. Instead, we demonstrate a typical OpenBR work flow by training, executing, and evaluating the classic Eigenfaces [22] algorithm. This example is available in the repository as `scripts/helloWorld.sh`.

Figure 3 shows how to train a face recognition algorithm. An *algorithm* is a method for enrolling a *template* associated with a method for comparing two templates. Training requires specifying what data to train on and where to save the learned *model*. By convention, model files have no extension.

Figure 4 uses the trained `Eigenfaces` model to enroll and compare images. The *path* informs the system where images can be found. *Compare* requires three parameters, *target gallery*, *query gallery*, and *output*, where target and query galleries form the columns and rows of the output score matrix. This particular example uses file formats defined in the NIST Face Recognition Grand Challenge [18]. The `.xml` files are *Signature Sets* that specify the input images, and `.mtx` is a binary *Similarity Matrix*.

Figure 5 evaluates the accuracy of `Eigenfaces` using the scores from `scores.mtx`. *Eval* takes two parameters, a score matrix as input and a `.csv` file of statistics as output. *Plot* takes one or more computed statistics files and produces a multi-page report with figures including ROCs, CMCs, and score distributions.

```
$ br -eval scores.mtx results.csv -plot
    results.csv results.pdf
```

Figure 5: Generating ROC, CMC, and other figures from the similarity matrix produced in Figure 4.

3. Architecture

This section introduces the main software abstractions intended for developers interested in extending OpenBR. There are two primary data structures in the framework and six abstract interfaces which constitute the plugin system. With the exception of a small core library, all OpenBR functionality is provided by *plugins*, classes that implement an abstract interface and declare themselves with the `BR_REGISTER` macro.

3.1. Data Structures

The OpenBR API is written at a high level, where input and output arguments are generally files on disk. The `File` struct is used to represent these arguments, storing the file path and a key/value table of associated metadata. A file's extension plays a particularly important role; it is used to identify which plugin should interpret the file. For example, during enrollment an `.xml` file is parsed by the `xmlGallery` plugin that treats the file as a NIST XML signature set.

A `Template` is biometric data, represented using OpenCV matrices, with an associated `File`. While templates tend to have only one matrix, experience has shown that it is convenient for a template to own a list of matrices in order to implement certain image processing transformations that either expect or produce multiple matrices.

In summary, files generally constitute the inputs and outputs of the API, and the plugin system relies on a file's extension to determine how to parse it. Templates represent the file and its data as it is enrolled and compared, and serve as the inputs and outputs of OpenBR's embedded image recognition language. `FileList` and `TemplateList` are provided as convenience classes for operating on lists of files and templates.

3.2. Plugins

Plugins are the preferred means of adding new functionality to OpenBR. For most researchers, they are likely the only classes that need to be designed in order to implement and evaluate a new algorithm. Extending OpenBR is simply a matter of adding plugin source code to `openbr/plugins` and recompiling. One particular virtue of the framework is that plugin classes need not provide header files, as every plugin implementation inherits from a carefully designed abstract interface.

There are six abstract interfaces in total, allowing OpenBR capability to be extended in a variety of ways. The `Format`, `Gallery`, and `Output` interfaces are used to implement new file formats. `Transform` and `Distance` plugins are the mechanism for inventing new techniques for template enrollment and comparison. Lastly, the `Initializer` interface enables allocation and deallocation of static resources at the beginning and the end of program execution.

A `Format` represents a template on disk either before or after enrollment. For example, images, videos, MATLAB matricies, and many other extensions can be interpreted by format plugins.

A `Gallery` represents a template list on disk either before or after enrollment. The NIST `.xml` signature set and OpenBR binary `.gal` are the standard plugins for storing template lists before and after enrollment, though many others exist including Weka `.arff`.

An `Output` represents the result of comparing two galleries. The NIST `.mtx` binary similarity matrix is the preferred output, though many others exist including `.rr` rank retrieval and `.csv` plain text score matrix.

A `Transform` is a single step in a template generation algorithm, it applies the same image processing or numerical analysis algorithm to every template it receives. Transforms can be either trainable (e.g., `LDA`) or untrainable (e.g., `LBP`). Time-varying transforms also exist to support object tracking in video.

A `Distance` is capable of comparing two templates and returning a similarity score. OpenBR supports many common similarity metrics including norm-based, cosine, Chi-squared, and Bhattacharyya. Section 4.5 discusses a particular distance metric novel to OpenBR.

Commercial algorithms can also be added to OpenBR by wrapping them in `Transform` and `Distance` plugins. To date, six commercial systems have been leveraged through the OpenBR API.

4. Face Recognition

While algorithms implemented within the OpenBR project are applicable to many biometric disciplines, particular effort has been devoted to the scenario of facial recognition. The default face recognition algorithm in OpenBR is based on the Spectrally Sampled Structural Subspaces Features (4SF) algorithm [11]. 4SF is a statistical learning-based algorithm used previously to study the impact of demographics [12] and aging [13] on face recognition performance.

The 4SF algorithm is not claimed to be superior to other techniques in the literature, instead it is representative of modern face recognition algorithms in its use of face representations and feature extraction. As will be shown, OpenBR's implementation of 4SF yields accuracies comparable to some commercial face recognition systems. Furthermore, the 4SF algorithm demonstrates strong accuracy improvements through statistical learning, allowing OpenBR to differentiate itself from commercial systems in its ability to be trained on specific matching problems like heterogeneous face recognition. This section discusses the 4SF algorithm in OpenBR, following the principal steps outlined in Figure 1.

4.1. Detection

OpenBR wraps the OpenCV Viola-Jones object detector [23] and offers frontal face detection with the syntax `Cascade(FrontalFace)`. For eye detection, a custom C++ port of the ASEF eye detector [3] is included in OpenBR as `ASEFEyes`.

4.2. Normalization

Faces are registered using the detected eye locations to perform a rigid rotation and scaling via the `Affine(...)` transform. For experiments in this paper, faces are cropped to 128x128 pixels, with the eyes inset 35% from the sides and 25% from the top.

The face recognition algorithm follows the illumination preprocessing steps suggested by Tan and Triggs [21] when extracting local binary patterns. Namely, a Gaussian blur Blur(σ), a difference of Gaussians DoG(σ_1,σ_2), gamma correction Gamma(γ), and Contrast Equalization ContrastEq(α,τ).

4.3. Representation

The face recognition algorithm uses both $\text{LBP}^{u2}_{8,1}$ [1] and SIFT [15] descriptors sampled in a dense grid across the face. Histograms of local binary patterns are extracted in an 8x8 pixel sliding window with a 6 pixel step. One hundred SIFT descriptors are sampled from a 10x10 grid with a descriptor radius of 12 pixels. A PCA decomposition retaining 95% of the variance is learned for each local region, with descriptors then projected into their corresponding Eigenspace and normalized to unit L_2-norm.

4.4. Extraction

The next step is weighted spectral sampling, whereby all per-region feature vectors are concatenated and random sampling is performed weighted on the variance of each dimension. For this paper, twelve samples were extracted, each with dimensionality equal to 5% of the entire feature space. LDA is then applied on each random sample to learn subspace embeddings that improve the discriminability of the facial feature vectors.

Lastly, descriptors are once again concatenated together and normalized to unit L_1-norm. Consistent with observations in [5], we have found that a simple normalization of the feature vectors to unit L_1 or L_2-norm after subspace projection can substantially improve accuracy in many pattern recognition scenarios.

Distance Metric	Template Size (KB)	Comparisons / Second	Accuracy (%) @ FAR = 0.1%
L_1	3.00	3.4×10^5	88 ± 1
L_1^{byte}	0.75	4.3×10^5	88 ± 1
L_1^{byte} SSE	0.75	2.6×10^6	88 ± 1

Table 2: Comparison of L_1^{Byte} against the conventional L_1 distance. Quantizing feature vector dimensions to 8-bit integers reduces the template size four-fold with no loss in accuracy. Implementing with SSE instructions further improves the comparison speed.

4.5. Matching

OpenBR supports a purportedly novel matching strategy that achieves state of the art matching speeds with negligible impact on matching accuracy. Here, we introduce the L_1^{byte} distance metric which, given an algorithm that compares feature vectors using the L_1 distance, quantizes the vectors to 8-bit unsigned integers as the last step in template generation.

Implementing L_1^{byte} requires computing the global maximum v_{max} and minimum v_{min} scalar values across all dimensions of the final training feature vectors. Then, during enrollment, a feature vector **fv** is scaled to the interval $[0, 255]$ and casted to an unsigned 8-bit integer:

$$\mathbf{fv}' = \text{uint8\_t}\left(255 \cdot \frac{\mathbf{fv} - v_{min}}{v_{max} - v_{min}}\right) \quad (1)$$

This quantization step reduces template size four-fold by exploiting the observation that the IEEE floating point format provides more precision than necessary to represent a feature vector. We note that the strategy is likely only applicable to the L_1 distance, as other metrics generally require multiplication steps that are not representable with 8-bit precision.

OpenBR further improves matching speed by using the `_mm_sad_epu8` Streaming SIMD Extensions (SSE) instruction [9] available on modern *x86/x64* CPUs, which computes the sum of the absolute difference of two 16-dimension 8-bit vectors. Using this instruction in conjunction with the aforementioned quantization step improves the template comparison speed of our matcher by nearly an order of magnitude, allowing us to achieve *several million comparisons per CPU thread per second* (Table 2).

4.6. The Complete Algorithm

Combining all the steps introduced above yields the complete algorithm shown in Figure 6. The design of new

```
Open+Cvt(Gray)+Cascade(FrontalFace)+
    ASEFEyes+Affine(128,128,0.33,0.45)+(
    Grid(10,10)+SIFTDescriptor(12)+ByRow
    )/(Blur(1.1)+Gamma(0.2)+DoG(1,2)+
    ContrastEq(0.1,10)+LBP(1,2)+
    RectRegions(8,8,6,6)+Hist(59))+PCA
    (0.95)+Normalize(L2)+Dup(12)+
    RndSubspace(0.05,1)+LDA(0.98)+Cat+
    PCA(0.95)+Normalize(L1)+Quantize:
    NegativeLogPlusOne(ByteL1)
```

Figure 6: The complete and unambigious definition of the 4SF face recognition algorithm in OpenBR, expressed in the algorithm description language.

facial representations is one of the most active areas of face recognition research, and helps illustrate one of the many ways OpenBR and its algorithm description language can be used as a time saving resource by researchers. By designing new representations as OpenBR plugins, researchers can immediately compare them against common representations (e.g., LBP, SIFT, Gabor), combine them with various learning algorithms, plot different accuracy metrics, and perform all of these tasks in a reproducible and deployable manner.

5. Experiments

This section provides results from experiments on twelve benchmark face datasets designed to span a wide range of applications and image characteristics. Template size and accuracy of the OpenBR 4SF algorithm are compared against three commercial off-the-shelf face recognition systems.

5.1. Datasets

In the interest of brevity we have omitted descriptions of each individual dataset considered in this paper. Instead, references for the datasets are provided in Table 3, example images are shown in Table 4, and aggregate dataset statistics are available in Table 5.

5.2. Algorithms

Two versions of the previously described 4SF algorithm are considered, one using open source face alignment (OpenBR) and the other using commercial alignment (BR-A). Accuracy is compared against three commercial systems (A, B, C). We intentionally choose *not* to tune algorithm parameters for each dataset, instead the algorithm is trained on all datasets exactly as stated in Figure 6.

Key	Dataset
CUFS	CUHK Face Sketch Database [24]
CUFSF	CUHK Face Sketch FERET Database [25]
FB	FERET fa vs. fb partitions [19]
FC	FERET fa vs. fc partitions [19]
Dup1	FERET fa vs. dup1 partitions [19]
Dup2	FERET fa vs. dup2 partitions [19]
FRGC-1	Face Rec. Grand Challenge Exp. 1 [18]
FRGC-4	Face Rec. Grand Challenge Exp. 4 [18]
HFB	CASIA Heterogeneous Face Biometrics [14]
LFW	Labeled Faces in the Wild [8]
MEDS	The Multiple Encounter Dataset [6]
PCSO	Pinellas County Sheriffs Office mugshots

Table 3: Abbreviations used for each dataset.

Dataset	Subjects	Gallery Images	Probe Images
CUFS	606	606	606
CUFSF	1,194	1,194	1,194
FB	1,196	1,196	1,195
FC	1,196	1,196	194
Dup1	1,196	1,196	772
Dup2	1,196	1,196	234
FRGC-1	466	16,028	*
FRGC-4	466	16,028	8,014
HFB	100	400	400
LFW	5749	13,233	*
MEDS	518	1,216	*
PCSO	5,000	10,000	*

Table 5: Dataset statistics. *Gallery set used as probes with self-matches removed.

5.3. Results

One challenging aspect of reporting on this wide variety of datasets is the lack of a common training and testing protocol. In the interest of allowing comparison across datasets, we opt to use 5-fold cross validation and report true accept rates at a false accept rate of 0.1%.

Table 6 compares the template size generated for each algorithm. OpenBR has the smallest templates, though templates from commercial system B are of similar size. BR-A template size is not listed as it is not statistically different than OpenBR.

Table 7 compares the true accept rates of each algorithm. The commercial systems tend to outperform OpenBR on the classic face recognition datasets like FERET and FRGC. There is less of a discrepancy in performance on some of

Table 4: Example genuine match pairs from each dataset.

	Template Size (KB)			
Dataset	OpenBR	A	B	C
CUFS	0.55 ± 0.01	67 ± 17	2.8 ± 0.0	5.0 ± 0.0
CUFSF	1.10 ± 0.01	70 ± 20	2.8 ± 0.0	5.0 ± 0.0
FB	0.98 ± 0.01	70 ± 20	2.8 ± 0.0	5.0 ± 0.0
FC	0.48 ± 0.01	70 ± 20	2.8 ± 0.0	5.0 ± 0.0
Dup1	0.72 ± 0.04	70 ± 20	2.8 ± 0.0	5.0 ± 0.0
Dup2	0.52 ± 0.02	70 ± 20	2.8 ± 0.0	5.0 ± 0.0
FRGC-1	1.58 ± 0.03	65 ± 13	2.8 ± 0.0	5.0 ± 0.0
FRGC-4	1.65 ± 0.02	65 ± 13	2.8 ± 0.0	5.0 ± 0.0
HFB	0.19 ± 0.02	64 ± 11	2.8 ± 0.0	5.0 ± 0.0
LFW	2.09 ± 0.01	96 ± 28	2.8 ± 0.0	5.0 ± 0.0
MEDS	0.60 ± 0.04	78 ± 26	2.8 ± 0.0	5.0 ± 0.0
PCSO	2.02 ± 0.02	72 ± 22	2.8 ± 0.0	5.0 ± 0.0

Table 6: Average gallery template size (KB) across different face recognition algorithms and datasets. Five-fold cross validation uncertainty reported at one standard deviation.

	Accuracy (%) @ FAR = 0.1%				
Dataset	OpenBR	BR-A	A	B	C
CUFS	67 ± 5	86 ± 2	83 ± 3	72 ± 3	89 ± 4
CUFSF	33 ± 3	41 ± 3	32 ± 2	19 ± 3	42 ± 2
FB	94 ± 1	100 ± 1	100 ± 1	100 ± 0	100 ± 0
FC	94 ± 4	99 ± 2	100 ± 0	100 ± 0	100 ± 0
Dup1	76 ± 4	86 ± 2	93 ± 3	98 ± 1	100 ± 0
Dup2	67 ± 7	85 ± 7	95 ± 4	98 ± 3	99 ± 1
FRGC-1	89 ± 2	91 ± 1	96 ± 1	100 ± 0	100 ± 0
FRGC-4	38 ± 4	49 ± 4	58 ± 7	98 ± 1	99 ± 0
HFB	22 ± 5	29 ± 5	25 ± 8	68 ± 5	92 ± 3
LFW	12 ± 2	23 ± 4	45 ± 4	39 ± 5	54 ± 2
MEDS	56 ± 6	60 ± 7	59 ± 9	88 ± 3	97 ± 3
PCSO	82 ± 1	90 ± 1	81 ± 2	96 ± 1	98 ± 0

Table 7: Average true accept rate at a false accept rate of one-in-one-thousand across different face recognition algorithms and datasets. Five-fold cross validation uncertainty reported at one standard deviation.

the heterogeneous datasets like CUFS and CUFSF, which demonstrates the benefit of a system that can be retrained on a new problem. Nevertheless, algorithm C performs particularly well across all of the datasets. If these results are found to hold up on sequestered datasets, it would suggest that methods popular in the academic literature lag considerably behind the best proprietary algorithms.

5.4. FRVT 2012

The OpenBR algorithm was also submitted to NIST's Face Recognition Vendor Test (FRVT) 2012 [7] for independent evaluation. OpenBR is believed to be the first open source system to formally compete in the series. Though the 2012 test has at least five academic participants, to our knowledge the source code for these systems has not been made publicly available. In the NIST reports, the OpenBR submission is denoted with the letter 'K'.

As of the end of Phase 1 in March 2013, amongst 17 competitors in the Class A verification task, OpenBR ranks 13th with a TAR of 64.8% on mugshots and 14th with a TAR of 76.1% on visas, each at a FAR of 0.1%. OpenBR ranks 2nd in template generation speed with a median enrollment time below 0.1 seconds, and 3rd in template size at 0.75 KB. Template comparison speed is not available.

OpenBR 4SF features were also used to train a support vector machine to compete in the Class D gender and age estimation tasks. For gender estimation, amongst 7 competitors OpenBR ranked 2nd on mugshots with 92.8% accuracy and 2nd on visas with 85.0% accuracy. For age estimation, amongst 6 competitors OpenBR ranked 4th on mugshots with a RMS error of 9.9 years for males and 11.2 years for females, and 4th on visas with a RMS error of 12.8 years for males and 14.8 years for females.

6. Conclusion

In this paper we introduced the OpenBR collaboratory for biometrics research and algorithm development. We then discussed the 4SF face recognition algorithm implemented in OpenBR, which offers a competitive baseline for researchers when a commercial system is unavailable. OpenBR offers the ability to isolate components of a biometric recognition process, allowing researchers to focus on particular steps in an algorithm within the context of a reproducible and deployable system.

Acknowledgments

The authors would like to thank The MITRE Corporation for releasing OpenBR as free open source software, and recognize Emma Taborsky and Charles Otto for their ongoing commitment to the collaboratory. Software development at MITRE was funded by the MITRE Sponsored Research (MSR) program, and B. Klare's research was supported by the Noblis Sponsored Research (NSR) program.

References

[1] T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(12):2037–2041, 2006.

[2] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 1997.

[3] D. S. Bolme, B. A. Draper, and J. R. Beveridge. Average of synthetic exact filters. In *IEEE Conf. on Computer Vision and Pattern Recognition, 2009*, pages 2105–2112.

[4] G. Bradski. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[5] Z. Cao, Q. Yin, X. Tang, and J. Sun. Face recognition with learning-based descriptor. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2010*, pages 2707–2714.

[6] A. Founds, N. Orlans, G. Whiddon, and C. Watson. NIST Special Database 32 - Multiple Encounter Dataset II (MEDS-II). www.nist.gov/itl/iad/ig/sd32.cfm, 2011.

[7] P. Grother, G. Quinn, and M. Ngan. Face Recognition Vendor Test (FRVT) 2012. http://www.nist.gov/itl/iad/ig/frvt-2012.cfm, mar 2013.

[8] G. B. Huang, M. Mattar, T. Berg, E. Learned-Miller, et al. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, 2008.

[9] Intel. Intel architecture instruction set extensions programming reference. http://download-software.intel.com/sites/default/files/319433-014.pdf, 2012.

[10] O. Jesorsky, K. J. Kirchberg, and R. W. Frischholz. Robust face detection using the hausdorff distance. In *Audio-and video-based biometric person authentication*, pages 90–95. Springer, 2001.

[11] B. Klare. Spectrally sampled structural subspace features (4SF). In *Michigan State University Technical Report, MSU-CSE-11-16*, 2011.

[12] B. Klare, M. Burge, J. Klontz, R. Vorder Bruegge, and A. Jain. Face recognition performance: Role of demographic information. *IEEE Trans. on Information Forensics and Security*, 7(6):1789–1801, 2012.

[13] B. Klare and A. K. Jain. Face recognition across time lapse: On learning feature subspaces. In *IEEE International Joint Conf. on Biometrics (IJCB), 2011*, pages 1–8.

[14] S. Z. Li, Z. Lei, and M. Ao. The HFB face database for heterogeneous face biometrics research. In *IEEE Conf. on Computer Vision and Pattern Recognition Workshops, 2009*, pages 1–8.

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[16] Y. M. Lui, D. Bolme, P. Phillips, J. Beveridge, and B. Draper. Preliminary studies on the good, the bad, and the ugly face recognition challenge problem. In *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW), 2012*, pages 9–16.

[17] P. J. Phillips, J. R. Beveridge, B. A. Draper, G. Givens, A. J. O'Toole, D. S. Bolme, J. Dunlop, Y. M. Lui, H. Sahibzada, and S. Weimer. An introduction to the good, the bad, & the ugly face recognition challenge problem. In *IEEE International Conf. on Automatic Face & Gesture Recognition (FG) Workshops, 2011*, pages 346–353.

[18] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek. Overview of the face recognition grand challenge. In *IEEE Conf. on Computer Vision and Pattern Recognition, 2005*, volume 1, pages 947–954.

[19] P. J. Phillips, H. Moon, S. A. Rizvi, and P. J. Rauss. The FERET evaluation methodology for face-recognition algorithms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(10):1090–1104, 2000.

[20] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, F. Pereira, and C. E. Rasmussen. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.

[21] X. Tan and B. Triggs. Enhanced local texture feature sets for face recognition under difficult lighting conditions. *IEEE Trans. on Image Processing*, 19(6):1635–1650, 2010.

[22] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[23] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[24] X. Wang and X. Tang. Face photo-sketch synthesis and recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 31(11):1955–1967, 2009.

[25] W. Zhang, X. Wang, and X. Tang. Coupled information-theoretic encoding for face photo-sketch recognition. In *IEEE Conf. on Computer Vision and Pattern Recognition, 2011*, pages 513–520.