Continuously Reproducing Toolchains in Pattern Recognition and Machine Learning Experiments

Conference Paper · August 2017

CITATIONS
22

READS
116

6 authors, including:

André Anjos
Idiap Research Institute
141 PUBLICATIONS **7,158** CITATIONS
SEE PROFILE

Manuel Günther
University of Zurich
45 PUBLICATIONS **1,100** CITATIONS
SEE PROFILE

Pavel Korshunov
Idiap Research Institute
89 PUBLICATIONS **1,450** CITATIONS
SEE PROFILE

Amir Mohammadi
Idiap Research Institute
22 PUBLICATIONS **305** CITATIONS
SEE PROFILE

Continuously Reproducing Toolchains in Pattern Recognition and Machine Learning Experiments

A. Anjos
Idiap Research Institute
Martigny, Switzerland
andre.anjos@idiap.ch

M. Günther
Vision and Security Technology
University of Colorado
Colorado Springs, USA
mgunther@vast.uccs.edu

T. Pereira, P. Korshunov, A. Mohammadi, S. Marcel
Idiap Research Institute
Martigny, Switzerland
marcel@idiap.ch

Abstract

Pattern recognition and machine learning research work often contains experimental results on real-world data, which corroborates hypotheses and provides a canvas for the development and comparison of new ideas. Results, in this context, are typically summarized as a set of tables and figures, allowing the comparison of various methods, highlighting the advantages of the proposed ideas. Unfortunately, result reproducibility is often an overlooked feature of original research publications, competitions, or benchmark evaluations. The main reason for such a gap is the complexity on the development of software associated with these reports. Software frameworks are difficult to install, maintain, and distribute, while scientific experiments often consist of many steps and parameters that are difficult to report. The increasingly rising complexity of research challenges make it even more difficult to reproduce experiments and results. In this paper, we emphasize that a reproducible research work should be repeatable, shareable, extensible, and stable, and discuss important lessons we learned in creating, distributing, and maintaining software and data for reproducible research in pattern recognition and machine learning. We focus on a specific use-case of face recognition and describe in details how we can make the recognition experiments reproducible in practice.

1 Introduction

The popularity of machine learning, especially neural networks, has been growing exponentially in the recent years. This growth manifested in an explosive number of available machine learning software, datasets, models, and techniques. Every respectable software company or a university is now providing and maintaining a machine learning software suite, ranging from open-source tools, such as TensorFlow [1] by Google, Caffe [2] by UC Berkeley, and Torch [3] by Facebook, to business-oriented solutions, including Azure [4] by Microsoft and Watson platform [5] by IBM. Datasets are also growing in size and numbers, with regularly organized competitions that provide challenging databases. Examples include the NIST speaker recognition evaluations [6], with more than a thousand hours recordings of more than two thousand identities, the MegaFace [7, 8] face recognition challenge, with four million images of more than six hundred thousand identities, and generic object recognition dataset COCO [9], with more than two million instances of eighty different object categories.

Such abundance of tools and data has a positive impact on advances in research, allowing scientists to quickly test their hypotheses and setup experiments, however, it also increases the complexity of an experiment setup. A paper that was published eighty years ago, e.g., on Linear Discriminant Analysis (LDA) by Ronald Fisher [10], is a self-contained piece of knowledge, and even now, its results can be verified and reproduced with pen and paper. A machine learning paper today is no

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

more bound by the 10-page text limit, and often corresponds just to the *tip of an iceberg*, since the experiments and the results depend on various software, data, configuration options, operating systems, and even hardware. Therefore, a researcher needs to make an additional effort for such a paper to become reproducible, so others can reliably repeat and verify its results.

Making a paper reproducible is clearly a challenging task. Operating systems have new releases, libraries that software depends on bring constant updates, bugs can be found and need to be corrected, documentation needs corrections, etc. With time, software inevitably changes and that may eventually degrade the reproducibility of published scientific work. Hence, a systematic approach and a formalized software organization need to be established to achieve reproducibility that is stable over time and useful to others. A systematic approach is especially important when considering typical research groups sizes and the constant flow of personnel that share ideas and software leading to published articles. If tried, maintaining current and past published reports (tens, if not hundreds of papers) reproducible over time can prove to be a rather challenging task.

In this paper, we consider a published paper to be reproducible if it is:

- **Repeatable**: It should be possible to re-run experiments declared in a report and obtain the same results, given the same selection of data, software, hyper-parameters, and evaluation protocol.
- **Shareable**: It should be possible to share the material (data and code) with others, so they can repeat experiments declared in a report.
- **Extensible**: It should be 'easy' to implement new research directions or evaluate new conditions on existing experimental infrastructure.
- **Stable**: The repeatability through time should be guaranteed, on a best-effort basis.

We share our approaches and lessons learnt on how to organize the research work and ensure that the resulted papers can satisfy these properties of reproducibility. We highlight the practical issues of reproducibility related to reusable experimental toolchains, source version control system, packaging, continuous integration, unit testing and documentation. We first provide, in Section 2, a generic overview of how to structure research-enabled software frameworks and experimental toolchains for them to be continuously reproducible, extensible and shareable. Then, in Section 3, we focus on a use case example of face recognition and describe, in details, how we achieve long term reproducibility of the published work on face recognition. In Section 4, we conclude with our outlook on reproducible research.

2 Framework

Today's state-of-the-art experiments will inevitably become, at some point, yesterday's baselines. Hence, extensibility is an important aspect to consider when designing frameworks which are meant to be reusable. It is possible to identify a number of common components through most experiments in machine learning and pattern recognition. Figure 1 tries to summarize such components and their interconnection forming a *"workflow"*. Processing typically starts from loading data from available raw samples (e.g., images, audio, and videos), that is then fed into a series of processing steps, which we refer to as *"toolchain"*. Toolchains input raw data samples and output representations of phenomena in such samples, which relates to the characteristics one wants to model. For example, the output of a toolchain may indicate whether a photograph was taken outdoors or indoors, whether a biological sample contains a contaminant, or whether a person is singing a song. The modelled phenomena, therefore, depends on nature of the problem one is trying to model. The last stage of the pipeline corresponds to *"analysis"* (or evaluation) of results obtained by combining raw data with the toolchain components and statistical models. Analysis yields merit figures indicating that a method performs better than another given the same input conditions.

The input dataset is probably one of the most important aspects of experimentation, as one tries to create models that express properties on real-world phenomena captured through the raw samples in such sets. To avoid bias and improve comparability between different toolchains, it is of current use to publish, alongside the raw data, usage protocols that establish which portions of the data may be used for training, model tuning, and/or evaluation. In competitions and public challenges [11], evaluation protocols on datasets may also impose performance figures requiring participants to follow

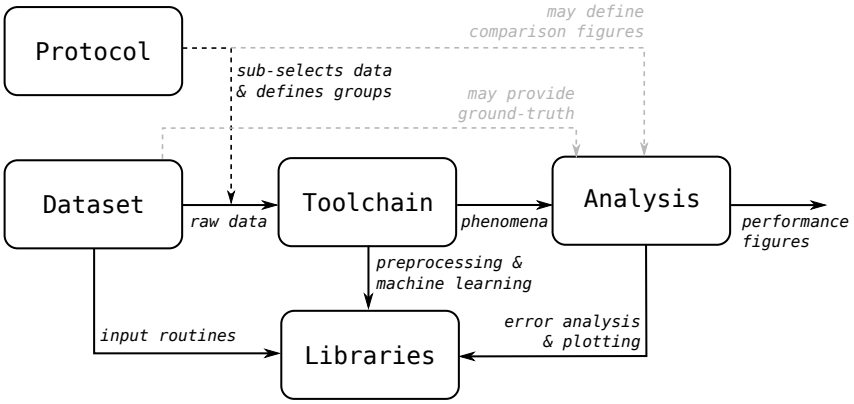


Figure 1: TYPICAL WORKFLOW IN MACHINE LEARNING AND PATTERN RECOGNITION. *Raw data from a dataset is read out from storage and input into a series of processing steps (toolchain) yielding information about detected phenomena. The final performance figures are extrapolated from these to provide markings for the system being tested. Read out from the database must often respect a usage protocol that, in some cases, also advise for standardized performance figures to facilitate comparison across different systems. Components to input and output data, pre-process, learn statistical models, and compute error figures are very frequently re-used and shared via libraries.*

strict evaluation procedures. Reading data from disk or databases may require access to input decoding routines, which are typically shared among different experiments (e.g., how to decode JPEG images or load electro-cardiogram data from a specific container format) through the use of external libraries, built and tested for such a purpose. Datasets may also provide labels or target values, allowing analysis modules to compare results obtained via learnt models to existing annotations.[1] Solutions exist in commercial products [11, 12] and for free [13] to load data through programmatic APIs, though none seem to tackle evaluation protocol integration.

Toolchains vary in complexity and size, but typically comprise (a) feature enhancement stage (e.g., data equalization, noise suppression, and subsampling); (b) feature extraction (e.g., local binary patterns, mel-frequency cepstral coefficients, and scale-invariant feature transforms), and (c) the creation of a statistical model allowing for the recognition and modelling of phenomena observed in raw data (e.g., support vector machines, neural networks, and gaussian mixtures). Not all toolchains are composed of the same number of processing steps. Their exact size and design depends on various factors including research directions, experience, and problem complexity. Currently, an abundant number of basic utilities for various machine learning fields can be used to build computing toolchains, providing their own advantages and downsides. Examples can be found in computer vision [14], audio processing [15], machine learning [1, 3], and so on. Toolchains for specific tasks are released from time to time as sub-products of conferences or journal articles [16, 17, 18], though very often, re-usability is not one of the features built-in, as connection to data readout, evaluation protocols, and analysis is frequently hard-coded.

Finally, the *analysis* stage computes figures of merit that help researchers to determine if workflows or specific combinations of hyper-parameters represent an improvement to established baselines or not. Figures of merit include, but not exclusively, measures of accuracy in detecting phenomena. The complexity of code or the modelling, execution time, or resource utilization may be also considered to be parameters of interest. In practice, dataset evaluation protocols often require specific figures of merit to be computed in the end of an experiment, allowing comparison across researchers and published material. In some domains, efforts go as far as standardizing evaluation techniques [19]. A great number of toolkits exist today for plotting [20, 21] and computing specific performance figures [22, 13]. Integration with existing workflows remain a manual task.

2.1 The *meta*-framework

Bare software frameworks are seldom successful in the long run without accompanying tools to manage various aspects related to its development and deployment. Allowing one self to readily repeat results obtained, as well as, transmitting such a knowledge to third parties are important

[1] Sometimes also referred to as ground-truth, even if the term may not always apply.

aspects in research and have a direct connection with reproducibility. We axe our analysis in four key tools which, in our experience, are essential elements to achieve continuable reproducibility and ease of sharing: (a) version control, (b) unit-testing and continuous quality control, (c) packaging and deployment and, finally, (d) documentation.

Source version control (VC) is a mechanism to keep track of the changes that is applied to files in a computer. VC accumulates the changes applied to the files in its history and changes can be reverted back to a particular point in history. VC gives software framework authors many advantages: it is the very first place where software revisions are defined, e.g., *version shipped to conference X*, which readily allows one to track where new bugs are introduced in the software. Existing VC systems often also accommodate tools boosting collaborative usage, issue reporting and tracking, authorization and authentication, which are equally important features. Since research reproducibility in machine learning and pattern recognition very frequently depends on large software stacks and configuration, we advocate precise version control of source-code.

Because of increasingly large software stacks, continuous reproducibility may be severely affected by the likewise continuous change in the underlying software dependencies and defaults. It is therefore equally essential to establish test routines which verify result stability, through the stipulated availability of a given scientific report and associated experiments. Test routines should be as thorough as possible and encompass basic tools, middleware ,and should reach, if possible, as far as experimental result checking. The re-use of existing software frameworks which are, in turn, properly tested may relief one from implementing basic testing and skip to middle and high-level software testing directly.

To ease test running and improve on the life expectancy of software packages, test suites are typically run in automated frameworks each time new software or configuration revisions are stored. This continuous quality control shall be flexible enough to allow its users to not only determine the impact of changes done to a given experiment, but also the impact caused by changes to **any** underlying software which may, one way or another alter original conclusions. When changes are detected, the use of VC associated to existing test suites must be carried on to determine the root causes of issues so to invalidate or confirm published hypothesis.

Deploying software stacks for one self or others is probably one of the biggest annoyances in attempts to reproduce experiments or re-use code into other research ventures in this field. The actual software stack required to run experiments may be composed of tens if not hundreds of software packages which must be compiled with a given version and in a particular way so reproduction can be achieved. It is, therefore, required of researchers seeking to reproduce results to learn how to properly handle and compile software, fix and report bugs, or solve issues associated to these activities.

Nevertheless, our experience in this area suggests that more interest and visibility can be drawn to one's research, if a complete software stack is easier to deploy, by-passing operating system requirements, compilation aspects and proper version handling. Simplified deployment instructions are possible by properly implementing software packaging using suitable tools. After software is properly packaged in compiled form, re-deployment for one self or third parties becomes very simple and can encourage collaboration and re-use which, in turn, directly impact research reproducibility and visibility. Packaging can optionally be hooked into continuous quality control, allowing contributions to be built, tested, and packaged in a single controlled framework.

Finally, it is difficult to envisage re-use, extensibility, and meaningful sharing of undocumented software frameworks. Properly documented code, in our experience, boosts reproducibility and share-ability, which finally impacts research visibility [23]. Documenting experimental research allows one to be able to re-run experiments, remember choices, and transmit those to newcomers or third-parties in an organized manner. Documentation generation may also be subject to continuous quality control, if its source code is kept alongside software. Some programming languages provide documentation standards allowing for both in-code and external sections to be combined creating very flexible documentation systems for a variety of needs.

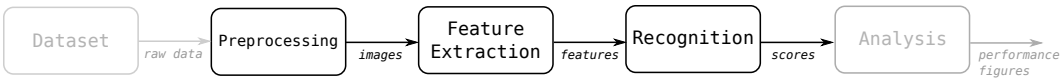


Figure 2: FACE RECOGNITION TOOLCHAIN. *The face recognition toolchain is split into several stages. These are: data preprocessing, feature extraction, and face recognition, including model enrollment scoring. Which images to use in which stage is defined by the protocol of the biometric database.*

3 Use-case in Biometrics: Bob

In form of a small use case, we describe a practical approach on how to make use of a framework, as described in Section 2, to run reproducible face recognition experiments. To do so, we created a reproducible source code package[2] that compares two state-of-the-art face recognition algorithms on the publicly available MOBIO dataset [24, 25], using the evaluation protocol `male`.

To run the experiments, we use a Python-based *biometrics framework*[3] that is part of Bob [26, 27, 28]. This framework implements a biometric recognition toolchain, which is depicted in Figure 2, as well as database access and evaluation tools. The implemented toolchain is divided into three stages, i) a preprocessing stage for data enhancement, ii) a feature extraction step to extract meaningful features from the raw images, iii) and a recognition stage, which is composed of the biometric-specific steps of model enrollment and the comparison of enrolled models with probe features to produce the final scores. Each stage of the toolchain is implemented in a class and has a defined input and output interfaces. The framework also controls the toolchain execution and can parallelize it to reduce running time. Using this framework, one can completely focus on the implementation of core algorithms, without being required to deal with evaluation protocols, data pipelining, dependencies between stages, or toolchain execution.

When implementing research that should be reproducible, we always follow a simple principle: one report leads to one software package requiring one execution environment. In our example, the environment is implemented with the Conda[4] package manager, which is used to install *all* the *specific* versions of required software on *any* Linux-based machine – usually conda allows for other operating systems, but dependent packages enforce some limitations. The fact that the specific versions of the software libraries are listed guarantees *stable* reproduction of the results over time. To make it possible for ourselves and others to re-install and use this package,[2] proper documentation of the installation procedure and commands to execute the experiments is vital and is included with the package.

Fortunately, Bob's biometrics framework[3] already comes with several implementations for database interfaces, preprocessors, feature extractors, and recognition algorithms. Our first algorithm that we test is the inter-session variability (ISV) modeling [29] of discrete cosine transform (DCT) features extracted from blocks of images [30]. As preprocessing, images are aligned using the eye locations provided by the database, and the face is cropped to photometrically enhanced 64×80 pixel images, while enrollment computes a Gaussian mixture model (GMM) in an ISV subspace of DCT block features. For this experiment, all tools are already implemented in the framework, so we only need to specify the parameters as defined in [30]. Running and *repeating* the experiment is as simple as calling a simple command line, please refer to our software package[2] for details. Due to the design of the framework, to *share* the experimental setup with others is as simple as sharing the parameters.

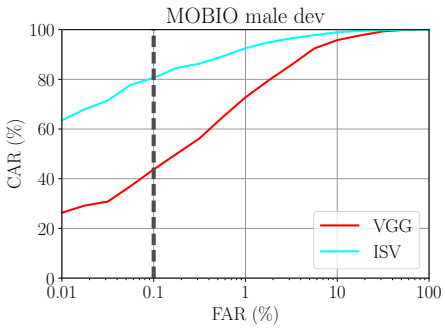
The second experiment will employ the Caffe [31] version of the publicly available VGG Face descriptor network [32]. Consonant with [32], we align images to 224×224 pixel color images and extract VGG Face descriptors from the images. The features of the enrollment images of the dataset are averaged for each subject, which are then compared using the cosine similarity measure. Here, the framework does not contain source code to extract features from a Caffe-based networks, so we have to *extend* it by writing a shallow wrapper[3] to connect the Caffe framework with the biometrics toolchain. Finally, all parameters are provided, which allows us (and anyone else) to run the second experiment.

After running both experiments on the same MOBIO image dataset, we evaluate the experiments using two typical performance measures for face recognition, both of which are proposed by the MOBIO dataset [24]. The resulting receiver operating characteristic (ROC) curve comparing the

[2] `https://gitlab.idiap.ch/bob/bob.paper.icml2017`

[3] `https://pypi.python.org/pypi/bob.bio.base`

[4] `https://conda.io/`



(a) ROC

Algorithm	EER	HTER
VGG	6.548 %	5.136 %
ISV	3.294 %	7.256 %

(b) EER/HTER

Figure 3: RESULTS. *The ROC curve on the development set as well as EER (development set) and HTER (evaluation set) for the two evaluated algorithms are presented.*

two algorithms on the development set are shown in Fig. 3(a), while equal error rate (EER) and half total error rate (HTER) are presented in Fig. 3(b).

The final work to do is to package the example, provide unit tests for the implementation of our source code, and incorporate it into the automatic testing framework, e.g., our continuous integration system that is built into the git source code distribution platform.[2] Running the test cases after installation also allows the user to verify that the installation process succeeded. Issues and questions may be posed via a publicly-accessible mailing list[5] connecting authors and users of Bob packages.

4 Conclusion

To guarantee long-time reproducibility, scientific work must be repeatable, shareable, extensible and stable over time. Committing to continuous reproducibility implies the creation and maintenance of a re-usable framework, as well as, addressing practical aspects such as version control, quality assurance, distribution and documentation. In our experience, efforts in making work reproducible can be reduced if, instead of working individually, researchers are unified into standardized frameworks and share common tools and environments. If the requirement for reproducibility is defined as a base objective in a research group, individuals will, in both short and longer terms, benefit from each other's help and be able to share work and experience in much more impactful ways than what is possible today.

We aim to follow these principles in our machine learning toolbox Bob and its existing reproducible research frameworks. Packages are maintained at our gitlab instance where we have implemented a quality assurance framework and documentation hosting. Experience to maintain and implement features into the framework are shared among students, postdocs, and researchers in a continuous turning cycle favoring exchange of ideas which are, since their inception, reproducible. External collaboration is welcome and maintained through a public mailing list.[5]

Because of the rising complexity of research results in the field, we expect to see an increase in the number of software frameworks tackling reproducible research in machine learning and pattern recognition for the coming years. The coupling of social networking with these frameworks may provide an opportunity to develop research tools, which can be easily shared relieving requirements for distribution, documentation, and deployment. The BEAT Platform [33] is one such example. Beyond social research, platforms such as this can also solve data distribution issues in a world with increasing dataset sizes, while implementing reproducibility assurance and honoring data privacy requirements.

5 Acknowledgements

The research and development required by the platform and leading to this article has received funding from the European Community's FP7 under the grant agreement 284989 (BEAT) and the Norwegian project SWAN.

[5] `https://www.idiap.ch/software/bob/discuss`

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[2] Y. Jia and E. Shelhamer, "Deep learning framework by bair." `http://caffe.berkeleyvision.org/`. Accessed: 2017-06-02.

[3] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[4] "Machine learning in microsoft azure." `https://azure.microsoft.com/en-us/services/machine-learning`. Accessed: 2017-06-02.

[5] "Ibm watson." `https://www.ibm.com/watson/`. Accessed: 2017-06-16.

[6] "Nist speaker recognition evaluation (sre)." `https://www.nist.gov/itl/iad/mig/speaker-recognition`. Accessed: 2017-06-02.

[7] I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller, and E. Brossard, "The megaface benchmark: 1 million faces for recognition at scale," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4873–4882, 2016.

[8] A. Nech and I. Kemelmacher-Shlizerman, "Level playing field for million scale face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[9] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[10] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[11] "Kaggle: The home of data science." `https://www.kaggle.com/`. Accessed: 2017-06-02.

[12] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] Itseez, "Open source computer vision library." `https://github.com/itseez/opencv`, 2015.

[15] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE Signal Processing Society, Dec. 2011. IEEE Catalog No.: CFP11SRW-USB.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[17] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," tech. rep., CMU-CS-16-118, CMU School of Computer Science, 2016.

[18] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *CVPR*, 2017.

[19] "Biometric performance testing and reporting – Part 6: Testing methodologies for operational evaluation," standard, International Organization for Standardization, Geneva, CH, Feb. 2012.

[20] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[21] T. Williams, C. Kelley, and many others, "Gnuplot 4.6: an interactive plotting program." `http://gnuplot.sourceforge.net/`, April 2013.

[22] N. I. of Standards and Technology, "Detector error-tradeoff (det) curve plotting software." `https://www.nist.gov/file/65991`, 2017.

[23] P. Vandewalle, "Code sharing is associated with research impact in image processing," *IEEE Computing in Science and Engineering*, vol. 14, pp. 42–47, July 2012.

[24] C. McCool, S. Marcel, A. Hadid, M. Pietikainen, P. Matejka, J. Cernocky, N. Poh, J. Kittler, A. Larcher, C. Levy, D. Matrouf, J.-F. Bonastre, P. Tresadern, and T. Cootes, "Bi-modal person recognition on a mobile phone: Using mobile phone data," in *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on*, pp. 635–640, July 2012.

[25] E. Khoury, L. El Shafey, C. McCool, M. Günther, and S. Marcel, "Bi-modal biometric authentication on mobile phones in challenging conditions," *Image and Vision Computing*, vol. 32, no. 12, pp. 1147–1160, 2014.

[26] A. Anjos, L. E. Shafey, R. Wallace, M. Günther, C. McCool, and S. Marcel, "Bob: a free signal processing and machine learning toolbox for researchers," in *20th ACM Conference on Multimedia Systems (ACMMM), Nara, Japan*, pp. 1449–1452, Oct. 2012.

[27] M. Günther, R. Wallace, and S. Marcel, "An open source framework for standardized comparisons of face recognition algorithms," in *Computer Vision - ECCV 2012. Workshops and Demonstrations* (A. Fusiello, V. Murino, and R. Cucchiara, eds.), vol. 7585 of *Lecture Notes in Computer Science*, pp. 547–556, Idiap Research Institute, Springer Berlin, Oct. 2012.

[28] M. Günther, L. El Shafey, and S. Marcel, *Face Recognition Across the Imaging Spectrum*, ch. Face Recognition in Challenging Environments: An Experimental and Reproducible Research Survey. Springer, 1 ed., Feb. 2016.

[29] R. Vogt and S. Sridharan, "Explicit modelling of session variability for speaker verification," *Comput. Speech Lang.*, vol. 22, pp. 17–38, Jan. 2008.

[30] R. Wallace, M. McLaren, C. McCool, and S. Marcel, "Inter-session variability modelling and joint factor analysis for face authentication," in *International Joint Conference on Biometrics*, 2011.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *International Conference on Multimedia (ACMMM)*, ACM, 2014.

[32] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *British Machine Vision Conference*, 2015.

[33] A. Anjos, L. El-Shafey, and S. Marcel, "BEAT: An Open-Source Web-Based Open-Science Platform," *ArXiv e-prints*, Apr. 2017.