Bob: A Free Signal Processing and Machine Learning Toolbox for Researchers

André Anjos, Laurent El Shafey, Roy Wallace, Manuel Günther, Chris McCool, Sébastien Marcel
Idiap Research Institute
Centre du Parc, rue Marconi 19, PO Box 592
CH-1920, Martigny, Switzerland
{andre.anjos,laurent.el-shafey,roy.wallace,manuel.guenther,christopher.mccool, sebastien.marcel}@idiap.ch

ABSTRACT

Bob is a free signal processing and machine learning toolbox originally developed by the Biometrics group at Idiap Research Institute, Switzerland. The toolbox is designed to meet the needs of researchers by reducing development time and efficiently processing data. Firstly, Bob provides a researcher-friendly Python environment for rapid development. Secondly, efficient processing of large amounts of multimedia data is provided by fast C++ implementations of identified bottlenecks. The Python environment is integrated seamlessly with the C++ library, which ensures the library is easy to use and extensible. Thirdly, Bob supports reproducible research through its integrated experimental protocols for several databases. Finally, a strong emphasis is placed on code clarity, documentation, and thorough unit testing. Bob is thus an attractive resource for researchers due to this unique combination of ease of use, efficiency, extensibility and transparency. Bob is an open-source library and an ongoing community effort.

Categories and Subject Descriptors

I.5 [**Pattern Recognition**]: General; I.4 [**Image Processing and Computer Vision**]: General

General Terms

Algorithms

Keywords

Open Source, Signal Processing, Machine Learning, Computer Vision, Pattern Recognition, Biometrics

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MM'12, October 29–November 2, 2012, Nara, Japan.
Copyright 2012 ACM 978-1-4503-1089-5/12/10 ...$15.00.

1. INTRODUCTION

The software requirements of researchers are quite unique. Firstly, to perform effective research requires a software development environment that allows for rapid prototyping and testing of experimental ideas. Yet at the same time, the implementation of the software must be fast enough to allow the researcher to run experiments on massive amounts of data. This is especially true in research covering the fields of multimedia and its applications. Finally, for research, the code needs to be clear and self-explanatory, with minimal dependence on other libraries that obfuscate the details of the implementation.

This paper presents *Bob*, a free signal processing (SP) and machine learning (ML) toolbox that is designed to meet the needs of researchers described above. Bob provides a researcher-friendly Python environment for rapid development, yet remains efficient for processing large amounts of multimedia data through the use of fast C++ implementations of identified bottlenecks. Bob is designed with reproducible research in mind, and currently provides an API to easily query and interface with several database protocols. In particular, several protocols for well-known biometric databases are integrated with the aim of improving the reproducibility and comparability of scientific publications. A strong emphasis is placed on code clarity, documentation, and thorough unit testing, with readability and maintainability always preferred over aggressive optimisation.

Bob has a broad scope in terms of SP that is intended to cover both computer vision and audio processing. In ML, it already includes many tools for dimensionality reduction, clustering, generative modelling, and discriminative classification. Most importantly, Bob is designed to be extensible. Specifically, the Bob library makes it easy for researchers to prototype their ideas and algorithms in a user-friendly Python environment, then later port easily to C++ for speed, all the while operating within the scope of the Bob library. Bob includes a clean Python environment and clean C++ environment, with transparent interaction facilitated by a thin layer that the researcher does not see and does not have to worry about. The researcher can thus quickly develop an idea in Python while seamlessly exploiting the fast C++ codebase provided by Bob. This demonstrates the great power of Bob, that new combinations and variations of feature types and algorithms can be implemented and tested quickly.

By supporting the development needs of researchers

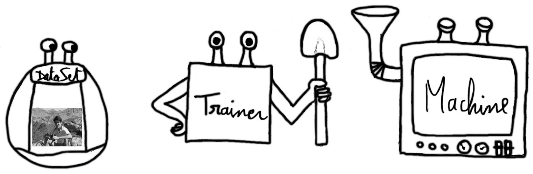

Figure 1: Original concepts from Torch: DataSet, Trainer and Machine. the Trainer uses data from the DataSet to train a Machine.

through the use of a common open-source software framework, the vision of Bob is to improve the reproducibility of publications across the multimedia research field into the future.

In Section 2 we highlight Bob's contributions with respect to prior software packages, followed by an overview of the Bob library in Section 3. The application of Bob to a simple pattern classification example is presented in Section 4. The extensibility of the Bob library is discussed in Section 5 before concluding with a description of future work in Section 6.

2. PRIOR WORK

A number of packages developed for ML and/or SP are currently available. Among others, Java-ML [1], scikit-learn [2], Shark [3] and OpenCV [4] are among the most solid, well maintained examples. Yet none of these libraries provide a complete set of tools for managing all aspects of research experimentation, including database interfaces, alternative clean APIs for sped-up implementations, as well as scriptable plotting utilities. The OpenCV library is developed primarily in C++ and unfortunately lacks dataset APIs or integrated analysis utilities. Java-ML and Shark suffer from similar problems. Among all choices, scikit-learn is possibly the closest competitor to Bob. It provides dataset APIs and ML algorithms, but lacks basic SP functionality and a clean C++ API for eventually speeding up identified bottlenecks. Users of those platforms will be inevitably confronted with either a lack of a convenient programming environment for research, or speed problems.

Specifically in computer vision, many libraries are already available such as VXL [5], RAVL [6], OpenCV [4], Torch3vision [7][1], and others. Of these, Torch3vision is particularly notable for its use of a unified and modular framework based on the prior work of the ML library Torch3 [8]. A particularity of Torch3 is its specific conceptual design (Fig. 1) of ML algorithms as a DataSet, Machine or Trainer. This enabled a modular implementation of ML algorithms that are essential tools for most pattern recognition tasks such as object detection or recognition in images, but also speech recognition in audio signals for instance. Torch3vision was later proposed to augment Torch3 with functionalities to manipulate and process images directly with ML algorithms.

More recently, Torch3 has diverged into two different evolutions, Torch5 [9] and finally Torch7 [10] using a Matlab-like environment powered by the lua [11] programming language. Unfortunately, a large proportion of the most popular ML algorithms from Torch3 have not been ported to Torch5 nor Torch7, and the original conceptual design is not as strong. Furthermore, Torch5 and Torch7 are based on an unusual language (lua) that prevents them from taking advantage of recent scientific resources (e.g. SciPy, Python Imaging Library) that are all written in a widely adopted language: Python.

Bob is a library that (1) leverages on the strong original concept from Torch3 and its ML algorithms, (2) generalizes the image processing from Torch3vision to more general SP, (3) uses a simple yet effective programming language (Python) as a convenient development environment for researchers, (4) provides extensive documentation (user and developer guides) and (5) guarantees the quality of the source code to foster reproducible research.

[1]`http://torch3vision.idiap.ch/`

3. OVERVIEW

At the implementation side, Bob can be considered as a collection of tools and interfaces implemented in both the C++ and Python languages. When first introduced to Bob, users get in touch with its Python application programming interface (API). This approach allows a laboratory-like fast pace development with scriptable constructions and includes plotting, automatic name completion and built-in reference documentation. Hidden behind the Python API, programming savvy users will find a collection of C++ utilities. Constructions in C++ are exclusively used when developers wish to speed-up execution time of their Python code.

The fundamental data structure of Bob consists of multi-dimensional arrays. In SP and ML, arrays are a suitable representation for many different types of digital signals such as images, audio data or features of various kinds. At the C++ level, this support is achieved with Blitz++ [12], whereas at the Python level, NumPy [13] arrays are used. To exchange data between C++ and Python, Bob makes use of the Boost template libraries [14]. This allows transparent movement of data structures from C++ to Python and vice-versa, as efficiently as possible. It allows both the Python and the C++ code to be developed independent of each other and to be based on native constructions.

The code base is subdivided into packages. There is no notion of layering in the software architecture. Bob is actually composed of a number of re-usable components that can be deployed either separately or jointly depending on user requirements. The diagram in Figure 2 may help in understanding what is the (loose) inter-dependency of Bob‘s internal packages and external software. Optional packages and external dependencies are marked with dashed lines. Functionality shipped with binary builds depend on software availability during compilation.

Below we outline some of the major features currently provided by Bob's packages:

Math and signal processing (`math`; `sp`): ML algorithms and SP usually rely on a sequence of low level mathematical operations. For efficiency purposes, eigenvalue decomposition, matrix inversion and other linear algebra are available and implemented using LAPACK [15] routines at the C++ level. In addition, the Fast Fourier Transform is made available via a bridge to the FFTW library [16].

Image processing (`ip`): Numerous image processing tools are provided such as filtering (Gaussian, Median, Gabor), visual feature extraction (LBP, SIFT bridge to VLFeat [17]),

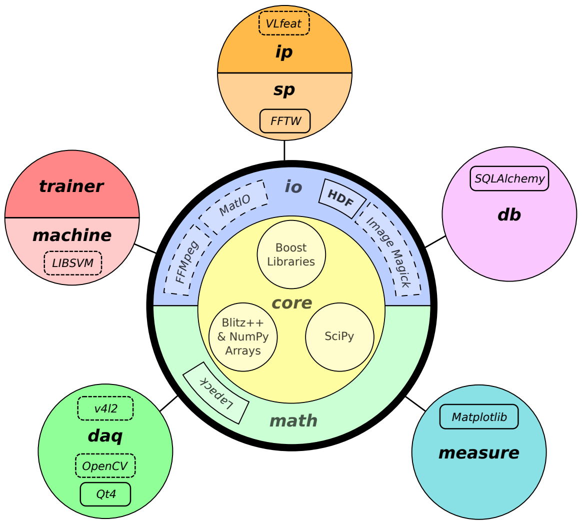

Figure 2: Internal software organization of Bob.

geometric normalization, illumination normalization and optical flow.

Machine learning (`machine`; `trainer`): Bob has been developed by researchers tackling many machine vision problems. Several ML algorithms have been integrated into the library. Dimensionality reduction is supported using Principal Component Analysis, Linear Discriminant Analysis and its probabilistic variant. There are data clustering algorithms such as k-means and classification is possible using both generative modeling techniques (Gaussian mixture models, Joint Factor Analysis) and discriminative approaches such as Multi-Layer Perceptrons or Support Vector Machines (via a LIBSVM [18] bridge).

Input and output (`io`): The library has been designed to run on various platforms and to be easily interfaced with any other software. We have chosen the open and portable HDF5 [19] library and file format as our core feature for storing and managing data. HDF5 is very flexible and hence allows us to store simple multi-dimensional arrays as well as complex ML models. Many tools for viewing, and analyzing the data are already available. In addition, we also support loading and storing most image formats thanks to ImageMagick [20], videos through FFmpeg [21] as well as standard Matlab file using MatIO [22].

Database support (`db`): The library currently provides an API to easily query and interface with database protocols. In particular, several protocols for well-known biometric databases are integrated with the aim at improving reproducibility and comparability of scientific publications.

Performance Evaluation (`measure`): A module of the library is dedicated to performance evaluation. This includes the computation of false alarm and false rejection rates, equal error rates as well as the generation of plots such as ROC, DET or EPC curves.

4. EXAMPLE

In this section, we briefly describe how Bob is applied to pattern recognition tasks in general and provide a very simple example for the reader.

For a typical pattern recognition task, the Bob processing flow chart consists of four main steps, as shown in Figure 3.

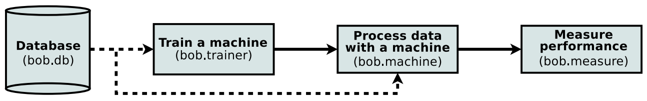
Database (bob.db)
Train a machine (bob.trainer)
Process data with a machine (bob.machine)
Measure performance (bob.measure)

Figure 3: A flow chart of the general processing chain for pattern recognition tasks in Bob.

The first step is the optional, but highly recommended, use of a database protocol. To facilitate reproducible research, each database protocol (in `bob.db`) specifies how the contents of a database should be used for experiments, such as what data can and cannot be used for training, development and performance evaluation. The second step is to train a machine (see `bob.trainer` and `bob.machine`), using the specified training data. The third step is to use the resulting machine to process a set of evaluation data. The fourth and final step is to measure the performance of the system (`bob.measure`), for example by reporting the classification error rate or creating a detection error tradeoff (DET) plot.

As an example of how to use Bob, we now describe the implementation of a simple one-versus-all classification task on the Iris flower data set[23][2]. In a Python environment, the first step is to import the Bob library, then load the database that is pre-defined in the `bob.db` package.

```
>>> import bob, numpy
>>> data = bob.db.iris.data()
```

In the second step we train a machine. In this example, we train a linear classifier using multi-class linear discriminant analysis (LDA). This is achieved by first creating the appropriate trainer (`bob.trainer.FisherLDATrainer`) that operates on the data and produces a linear classifier of type `bob.machine.LinearMachine`.

```
>>> trainer = bob.trainer.FisherLDATrainer()
>>> machine, e = trainer.train(data.values())
```

In the third step we use the machine to process the data by calling the `machine.forward` method for each sample in the database.

```
>>> out = {}
>>> for species in data.keys():
...   out[species] = machine.forward(data[species])
```

The fourth and final step is to measure the performance of the system output using the functions in `bob.measure`. For the sake of this example, assume we are interested in measuring the equal error rate (EER) when we classify *virginica* versus the other flower species (one-versus-all) using just the first LDA component. We first find the threshold, `t`, that separates the virginica (positive) samples from the other (negative) samples at the equal error rate (EER).

```
>>> pos = out['virginica'][:,0]
>>> neg = numpy.vstack([out['setosa'],
...                     out['versicolor']])[:,0]
>>> t   = bob.measure.eer_threshold(neg, pos)
```

Next, we calculate the false accept and reject rates at this threshold (which are equal in this case) and report the resulting EER.

[2]A more comprehensive overview of this example is given in the User's Guide of Bob, `http://www.idiap.ch/software/bob/`.

```
>>> far, frr = bob.measure.farfrr(neg, pos, t)
>>> print "EER: %.1f%%" % ((far+frr)/2*100)
```

The program thus produces the desired result: `EER: 2.0%`.

5. EXTENSIONS

The potential applications of Bob are wide and varied. While a significant set of functionality has been provided with the latest release of Bob, of course we could not cover all possibilities. It is important to note that Bob is extensible and provides a Python API for flexible development. Thus, the simplest way to expand Bob for your own usage is to write your own satellite package around Bob that, for instance, reproduces the experiments of your latest research. To encourage this and facilitate reproducible research, we have created a page on the Bob website dedicated to satellite packages[3]. An example of a satellite package is provided for the recent work presented in [24], another example shows how to build face verification systems using Bob and its database module, and further packages are expected shortly.

The second way to extend Bob is to join the Bob developer community and contribute to the expansion of Bob, as described in the detailed Developer's Guide[4]. After forking the Bob git repository, you can make your own building blocks for Bob. The structure is designed to make it easy to contribute your new functionality in a modular fashion. Finally, a request may be made for the new functionality to be incorporated into the official Bob repository. If the contribution is sufficiently documented and accompanied by unit testing it will likely be incorporated within the Bob project, ready for re-use by the rest of the research community.

6. CONCLUSIONS

This paper presented Bob, a free SP and ML toolbox originally developed at Idiap Research Institute and written in a mix of Python and C++. By supporting the development needs of researchers through the use of a common open-source software framework, the vision of Bob is to improve the reproducibility of publications across the multimedia research field into the future.

The Bob library will continue to be extended in several directions. Firstly, support for new databases (biometric and otherwise) will be added by implementing the corresponding experimental protocols. Secondly, the library will expand with the implementation of new tools for SP, ML and analysis. The addition of Windows support is also planned.

As Bob is an open-source library and an ongoing community effort, contributions are encouraged. We look forward to continuing to develop Bob, in the pursuit of faster turnaround on research software development and ever increasingly reproducible research.

7. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's FP7 under grant agreements 238803 (BBfor2), 257289 (TABULA RASA) and 284989 (BEAT). The authors would like to thank I. Chingovska and F. Moulin.

[3]`https://github.com/idiap/bob/wiki/Satellite-Packages`

[4]Access via `http://www.idiap.ch/software/bob/`

8. REFERENCES

[1] T. Abeel, Y. V. de Peer, and Y. Saeys, "Java-ML: A Machine Learning Library," *Journal of Machine Learning Research*, vol. 10, pp. 931–934, 2009.

[2] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python ," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] C. Igel, T. Glasmachers, and V. Heidrich-Meisner, "Shark," *Journal of Machine Learning Research*, vol. 9, pp. 993–996, 2008.

[4] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[5] "VXL." http://vxl.sourceforge.net, 2003.

[6] C. Galambos, "RAVL: Recognition And Vision Library." http://ravl.sourceforge.net, 2000.

[7] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of Torch," in *Proc. of the ACM Intl. Conf. on Multimedia*, pp. 1485–1488, 2010.

[8] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: A Modular Machine Learning Software Library," tech. rep., 2002.

[9] R. Collobert, "Torch5." http://torch5.sourceforge.net/, 2006.

[10] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A Matlab-like Environment for Machine Learning," *NIPS Workshop BigLearn*, 2011.

[11] R. Ierusalimschy, W. Celes, and L. de Figueiredo, "Lua." http://www.lua.org/, 1993.

[12] T. L. Veldhuizen, "Arrays in Blitz++," in *Proc. of the Intl. Scientific Computing in Object-Oriented Parallel Environments*, pp. 223–230, Springer-Verlag, 1998.

[13] T. E. Oliphant, "Python for Scientific Computing," *Computing in Science Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[14] R. Demming and D. Duffy, *Introduction to the Boost C++ Libraries; Volume I - Foundations*. No. v. 1, Datasim Education Bv, 2010.

[15] E. Anderson *et al.*, *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, third ed., 1999.

[16] M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[17] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms." http://www.vlfeat.org/, 2008.

[18] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, pp. 27:1–27:27, May 2011.

[19] The HDF Group, "Hierarchical Data Format, Version 5." http://www.hdfgroup.org/HDF5, 2000-2010.

[20] M. Still, *The Definitive Guide to ImageMagick*. Berkeley, CA, USA: Apress, 2005.

[21] "FFmpeg." http://ffmpeg.org/, 2012.

[22] "MatIO." http://matio.sourceforge.net/, 2012.

[23] R. A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, vol. 7, pp. 179–188, 1936.

[24] A. Anjos and S. Marcel, "Counter-Measures to Photo Attacks in Face Recognition: a public database and a baseline," in *Intl. Joint Conf. on Biometrics*, 2011.