Coelho, L P 2013 Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software* 1:e3, DOI: http://dx.doi.org/10.5334/jors.ac

SOFTWARE METAPAPER

Mahotas: Open source software for scriptable computer vision

Luis Pedro Coelho[1,2]

[1] Lane Center for Computational Biology, Carnegie Mellon University, Pittsburgh, United States of America

[2] Instituto de Medicina Molecular, Lisboa, Portugal

Mahotas is a computer vision library for Python. It contains traditional image processing functionality such as filtering and morphological operations as well as more modern computer vision functions for feature computation, including interest point detection and local descriptors.

The interface is in Python, a dynamic programming language, which is appropriate for fast development, but the algorithms are implemented in C++ and are tuned for speed. The library is designed to fit in with the scientific software ecosystem in this language and can leverage the existing infrastructure developed in that language.

Mahotas is released under a liberal open source license (MIT License) and is available from http://github.com/luispedro/mahotas and from the Python Package Index (http://pypi.python.org/pypi/mahotas). Tutorials and full API documentation are available online at http://mahotas.readthedocs.org/.

Keywords: computer vision, image processing

Funding statement
The author was supported by the Fundação para a Ciência e Tecnologia (grant SFRH/BD/37535/2007 to the author and grant PTDC/SAU-GMG/115652/2009 to Musa Mhlanga), by NIH grant GM078622 (to Robert F. Murphy), by a grant from the Scaife Foundation, by the HHMI Interfaces Initiative, and by a grant from the Siebel Scholars Foundation.

(1) Overview

Introduction

Mahotas is a computer vision library for the Python Programming Language (versions 2.5 and up, including version 3 and up). It operates on numpy arrays[1]. Therefore, it uses all the infrastructure built by that project for storing information and performing basic manipulations and computations. In particular, unlike libraries written in the C Language or in Java[2,3], Mahotas does not need to define a new image data structure, but uses the numpy array structure. Many basic manipulation functionality that would otherwise be part of a computer vision library are handled by numpy. For example, computing averages and other simple statistics, handling multi-channel images, converting between types (integer and floating point images are supported by mahotas) can all be performed with numpy builtin functionality. For the users, this has the additional advantage that they do not need to learn new interfaces.

Mahotas contains over 100 functions with functionality ranging from traditional image filtering and morphological operations to more modern wavelet decompositions and local feature computations. Additionally, by integrating into the Python numeric ecosystem, users can use other packages in a seamless way. In particular, mahotas does not implement any machine learning functionality, but rather advises the user to use another, specialised package, such as scikits-learn or milk.

Python is a natural “glue” language: it is easy to use state-of-the-art libraries written in multiple languages[4]. Mahotas itself is a mix of high-level Python and low-level C++. This achieves a good balance between speed and ease of implementation.

Version 1.0.2 of mahotas has been released recently and this is now a mature, well-tested package (the first versions were made available over 4 years ago, although the package was not named mahotas then). Mahotas runs and is used on different versions of Unix (including Linux, SunOS, and FreeBSD), Mac OS X, and Windows.

Implementation/architecture

Interface

The interface is a procedural interface, with no global state. All functions work independently of each other (there is code sharing at the implementation level, but this is hidden from the user). The main functionality is grouped into the following categories:

- **Surf:** Speeded-up Robust Features[5]. This includes both keypoint detection and descriptor computation.
- **Features:** Global feature descriptors. In particular, Haralick texture features[6], Zernike moments, local binary patterns[7], and threshold adjacency statistics (both the original[8] and the parameter-free versions[5]).
- **Wavelet:** Haar and Daubechies wavelets[10]. Forward and inverse transforms are supported.
- **Morphological functions:** Erosion and dilation, as well as some more complex operations built on these. There are both binary and grayscale implementations of these operators.
- **Watershed:** Seeded watershed and distance map transforms[11].
- **Filtering:** Gaussian filtering, edge finding, and general convolutions.
- **Polygon operations:** Convex hull, polygon drawing.

Numpy arrays contain data of a specific type, such unsigned 8 bit integer or floating point numbers. While natural colour images are typically 8 bits, scientific data is often larger (12 and 16 bit formats are common). Processing can generate floating point images. For example, a common normalization procedure is to subtract from each pixel location the overall pixel value mean; the result will be a floating point image even if the original image was integral. Mahotas works on all datatypes. This is performed without any extra memory copies. Mahotas is heavily optimised for both speed and memory usage (it can be used with very large arrays).

There are a few interface conventions which apply to many functions. When meaningful, a structuring element is used to define neighbourhoods or adjacency relationships (morphological functions, in particular, use this convention). Generally, the default is to use a cross as the default if no structuring filter is given.

When a new image is to be returned, functions take an argument named out where the output will be stored. This argument is often much more restricted in type. In particular, it must be a contiguous array. Since this is a performance feature (its purpose is to avoid extra memory allocation), it is natural that the interface is less flexible (accessing a contiguous array is much more efficient than a non-contiguous one).

Examples of Use

Code for this and other examples is present in the mahotas source distribution under the demos/ directory. In this example, we load an image, find SURF interest points, and compute descriptors.

We start by importing the necessary packages, including numpy and mahotas. We also use scipy.cluster, to demonstrate how the mahotas output can integrate with a machine learning package.

```
import numpy as np
import mahotas as mh
from mahotas.features import surf
from scipy.cluster import vq
```

The first step is to load the image and convert to 8 bit numbers. In this case, the conversion is done using standard numpy methods, namely astype. We use the function mahotas.demos.image_path to access a demonstration image.

```
import mahotas.demos
impath = mh.demos.image_path('luispedro.jpg')
f = mh.imread(impath, as_grey=True)
f = f.astype(np.uint8)
```

We can now compute SURF interest points and descriptors.

```
spoints = surf.surf(f, 4, 6, 2)
```

The surf.surf function returns both the descriptors and their meta data. We use numpy operations to retain only the descriptors (the meta data is in the first five positions):

```
descrs = spoints[:,6:]
```

Using scipy.cluster.vq, we cluster the descriptors into five groups. The function kmeans2 returns two values: the centroids, which we ignore; and the cluster ids, which we will use below to assign colours:

```
_,cids = vq.kmeans2(vq.whiten(descrs), 5)
```

Finally, we can show the points in different colours. In order to avoid a very cluttered image, we will only plot the first 64 regions.

```
colors = np.array([
   [ 255,  25,   1],
   [203,  77,  37],
   [151, 129,  56],
   [ 99, 181,  52],
   [ 47, 233,   5]])
f2 = surf.show_surf(f, spoints[:64], cids,
colors)
```

The show_surf only builds the image as a multi-channel (one for each colour) image. Using matplotlib [12], we finally display the image as Fig. 1.

```
colors = np.array([
[ 255,  25,   1],
[203,  77,  37],
[151, 129,  56],
[ 99, 181,  52],
[ 47, 233,   5]])
f2 = surf.show_surf(f, spoints[:64], cids,
colors)
```

The easy interaction with matplotlib is another way in which we benefit from the numpy-based ecosystem. Mahotas does not need to support interacting with a graphical system to display images.

Implementation

Mahotas is mostly written in C++, but this is completely hidden from the user as there are hand-written Python wrappers for all functions.The main reason that mahotas is implemented in C++ (and not in C, which is the language of the Python interpreter) is to use templates. Almost all C++ functionality is split across 2 functions:

1. A py_function which uses the Python C API to get arguments and check them.
2. A template function <dtype> which works for the type dtype performing the actual operation.

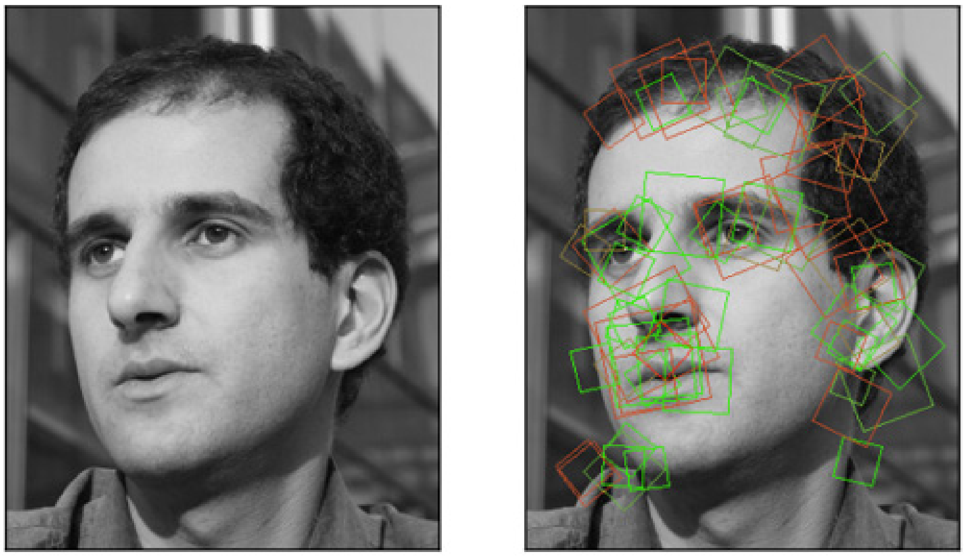

Fig. 1: Example of Usage. On the left, the original image is shown, while on the right SURF detections are represented as rectangles of different colours.

So, for example, this is how erode is implemented. py_erode consists mostly of boiler-plate code:

```
PyObject* py_erode(PyObject* self, PyObject*
args) {
   PyArrayObject* array;
   PyArrayObject* Bc;
   PyArrayObject* output;
   if (!PyArg_ParseTuple(args,”OOO”,&array,&Bc,&
output) ||
   !numpy::are_arrays(array, Bc, output) ||
   !numpy::same_shape(array, output) ||
   !numpy::equiv_typenums(array, Bc, output) ||
   PyArray_NDIM(array) != PyArray_NDIM(Bc)
   ) {
   PyErr_SetString(PyExc_RuntimeError,
TypeErrorMsg);
   return NULL;
   }
   holdref r_o(output);

#define HANDLE(type) \
   erode>type<(numpy::aligned_
array>type<(output), \
      numpy::aligned_array>type<(array), \
      numpy::aligned_array>type<(Bc));
   SAFE_SWITCH_ON_INTEGER_TYPES_OF(array);
#undef HANDLE
...
```

This functions retrieves the arguments, performs some sanity checks, performs a bit of initialization, and finally, switches in the input type with the help of the SAFE_SWITCH_ON_INTEGER_TYPES macro, which call the right specialisation of the template that does the actual work. In this example erode implements erosion:

```
template>typename T<
void erode(numpy::aligned_array>T< res
         numpy::aligned_array>T< array,
         numpy::aligned_array>T< Bc) {
      gil_release nogil;
      const int N = res.size();
      typename numpy::aligned_array>T<:iterator
                        iter = array.begin();
      filter_iterator>T< filter(array.raw_array(),
            Bc.raw_array(),
         ExtendNearest,
            is_bool(T()));
      const int N2 = filter.size();
      T* rpos = res.data();

      for (int i = 0;
         i != N;
            ++i, ++rpos, filter.iterate_
both(iter)) {
      T value = std::numeric_limits>T<::max();
      for (int j = 0; j != N2; ++j) {
         T arr_val = T();
         filter.retrieve(iter, j, arr_val);
         value = std::min>T<(value,
                        erode_sub(arr_val,
filter[j]));
      }
      *rpos = value;
   }
}
```

The template machinery makes the functions that use it very simple and easy to read. The only downside is that there is some expansion of code size when the compiler instantiates the function for the several integer and floating point types. Given the small size of these functions, the total size of the compiled library is reasonable (circa 6MiB on an Intel-based 64 bit system for the whole library).

In the snippet above, you can see some other C++ machinery:

- **gil_release:** This is a “resource-acquisition is object initialisation” (raii)4 object that releases the Python global interpreter lock (gil)5 in its constructor and gets it back in its destructor. Normally, the template function will release the gil after the Python-specific code is done. This allows several mahotas functions to run concurrently.
- **array:** This is a thin wrapper around PyArrayObject, the raw numpy data type, which has iterators which resemble the C++ standard library. It also handles

type-casting internally, making the code type-safer. This is also a raii object in terms of managing Python reference counts. In mahotas debug builds, this object additionally adds several checks to all the memory acesses.

- **filter_iterator:** This was adapted from code in the scipy.ndimage packages and it is useful to iterate over an image and use a centered filter around each pixel (it keeps track of all of the boundary conditions).

The inner loop is as direct an implementation of erosion as one would wish for: for each pixel in the image, look at its neighbours, subtract the filter value, and compute the minimum of this operation.

Efficiency

Operation	mahotas	pymorph	scikits-image	OpenCV
erode	1.6	15.1	7.4	0.4
dilate	1.5	9.1	7.3	0.4
open	3.2	24.3	14.8	NA
median filter (2)	226.9	NA	2034.0	NA
median filter (10)	2810.9	NA	1877.1	NA
center mass	5.0	NA	3611.2	NA
sobel	34.1	NA	62.5	6.2
cwatershed	174.8	58440.3	287.3	44.9
daubechies	18.8	NA	NA	NA
haralick	233.1	NA	7760.7	NA

Table 1: Efficiency Results for mahotas, pymorph, scikits-image, and openCV (through Python wrappers). Shown are values as multiples of the time that numpy.max(image) takes to compute the maximum pixel value in the image (all operations are over the same image). For scikits-image, features on the grey-scale cooccurrence matrix were used instead of Haralick features, which it does not support. In the case of median filter, the radius of the structuring element is shown in parentheses. NA stands for "Not Available."

Table 1 shows timings for different operations. These were normalized to multiples of the time it takes to go over the image and find its maximum pixel value (using the expression numpy.max(image)). The measurements shown were obtained on an Intel 64 bit system, running Ubuntu Linux. Due to the normalization, measurements obtained on another system (Intel 32 bits running Mac OS) were qualitatively similar.

The comparison is against Pymorph[13], which is a pure Python implementation of some of the same functions; scikits-image, which is a similar project to mahotas, but with a heavier emphasis on the use of Cython[14]; and OpenCV, which is a C++ library with automatically generated Python wrappers.

OpenCV is the fastest library, but this comes at the cost of some flexibility. Arguments to its functions must be of the exact expected type and it is possible to crash the interpreter if types do match the expected type (in the other libraries, including mahotas, all types are checked and an exception is generated which can be caught by user code). This is particularly relevant for interactive use as the user is often exploring and is willing to pay the speed cost of a few extra type checks to avoid a hard-crash.

Distribution and Installation

In keeping with the philosophy of blending in with the ecosystem, Mahotas uses the standard Python build machinery and distribution channels. Building and installing from source code is done using python setup.py install Alternatively, Python based package managers (such as easy_install or pip) can be used (mahotas works well with these systems).

For compiling from source, a C++ compiler is needed, as well as the development headers for Python and numpy.

There are binary packages available for Windows, maintained by Christoph Gohlke, and for FreeBSD and Linux Frugalware through their respective package systems.

Quality Control

Mahotas includes a complete automated suite of unit tests, which tests all functionality and include several regression tests. There are no known bugs in version 1.0.2. Occasional bugs discovered in previous released versions have been corrected before the next release.

The development is completely open-source and development versions are available. Many users have submitted bug reports and fixes.

(2) Availability

Operating system

Mahotas runs and is used on different versions of Unix (including Linux, SunOS, and FreeBSD), Mac OS X, and Windows.

Programming Language

Mahotas works in Python (minimal version is 2.5, but mahotas works with all more recent versions, including version in the Python 3 series).

Additional system requirements

None at runtime. Compilation from source requires a C++ compiler and the Python development headers.

Dependencies

At runtime, mahotas requires numpy to be present and installed.

List of contriubutors

Luis Pedro Coelho (Carnegie Mellon University and Instituto de Medicina Molecular), Zachary Pincus (Stanford University), Peter J. Verveer (European Molecular Biology Laboratory), Davis King (Northrop Grumman ES), Robert Webb (Carnegie Mellon University), Matthew Goodman (University of Texas at Austin), K.-Michael Aye (University of Bern), Rita Simões (University of Twente), Joe Kington

(University of Wisconsin), Christoph Gohlke (University of California, Irvine), Lukas Bossard (ETH Zurich), and Sandro Knauss (University of Bremen).

Code Repository

Name
Github

Persistent identifier
https://github.com/luispedro/mahotas

License
MIT

Publisher
Konstantin Nikolic

Date published
Since 2010 as mahotas. Some of the code had been previously made available under other names.

Archive

Name
PyPI

Persistent identifier
https://pypi.python.org/packages/source/m/mahotas/mahotas-1.0.2.tar.gz#md5=bc3e478a5deb0f2f05e9eb385bbb07ff
MD5: bc3e478a5deb0f2f05e9eb385bbb07ff

License
MIT

Date published
2013-05-04 (Mahotas v1.0.2)

(3) Reuse potential

Originally, this code was developed in the context of cellular image analysis. However, the code was designed so that mahotas would contain functionality that is not specific to cell image analysis and many computer vision pipelines can make use of it.

This package (or earlier versions of it) have been used by myself [15,16] and close collaborators in several publications[17]. Other groups have used it in published work in cell image analysis[18] and in other areas[19]. Ploshnik et al.[20]used mahotas to detect nanoparticles in electron microscopy images.

Mahotas provides many basic tools which can be combined to process images. It can be used in any problem which requires the processing of images to extract quantitative information.

Discussion

Python is an excellent language for scientific programming because of the inherent properties of the language and because of the infrastructure that has been built around the numpy project. Mahotas works in this environment to provide the user with image analysis and computer vision functionality.

Mahotas does not include machine learning related functionality, such as k-means clustering or classification methods. This is the result of an explicit design decision. Specialised machine learning packages for Python already exist[21,22,23,24]. A good classification system can benefit both computer vision users and others. As these projects all use Numpy arrays as their data types, it is easy to use functionality from the different project seamlessly (no copying of data is necessary).

Mahotas is implemented in C++, as the standard Python interpreter is too slow for a direct Python implementation. However, all of the Python interface code is handwritten, as opposed to using automatic interface generators like Swig[25]. This is more work initially, but the end result is of much higher quality, especially when it comes to giving useful error messages. When a type mismatch occurs, an automatic system will often be forced to resort to a generic message as it does not have any knowledge of what the arguments mean to the user. It will only know their automatically inferred types. A hand written system can also automatically convert arguments when meaningful and be more flexible without completely foregoing type checking.

Mahotas has been available in the Python Package Index since April 2010 and has been downloaded over fifty thousand times. This does not include any downloads from other sources. Mahotas includes a full test suite. There are no known bugs.

Acknowledgements
Mahotas includes code ported and incorporated from other projects. Initially, it was used in reproducing the functionality in the Subcellular Location Image Classifier (SLIC) tool from Robert F. Murphy's Lab[26] and the initial versions of mahotas were designed explicitly to support that functionality. The surf implementation is a port from the code from dlib,7 an excellent C++ library by Davis King. I also gleaned some insight into the implementation of these features from Christopher Evan's OpenSURF library and its documentation[27]. The code which interfaces with the FreeImage library, was written by Zachary Pincus and some of the support code was written by Peter J. Verveer for the scipy.ndimage project. All of these contributions were integrated while respecting the software licenses under which the original code had been released. Robert Webb, a summer student at Carnegie Mellon University, worked with me on the initial local binary patterns implementation. Finally, I thank the several users who have reported bugs, submitted fixes, and participated on the project mailing list.

Stéfan van der Walt and Andreas Müller offered helpful comments on a draft version of this manuscript.

References

1. **van der Walt S, Colbert S** and **Varoquaux G** 2011 The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering* 13(2): 22–30, DOI: http://dx.doi.org/10.1109/MCSE.2011.37
2. **Pietzsch T, Preibisch S, Tomančák P** and **Saalfeld S** 2012 Imglib2–generic image processing in java. *Bioinformatics* 28(22): 3009–3011. DOI: http://dx.doi.org/10.1093/bioinformatics/bts543

3. **Marcel S** and **Rodriguez Y** 2010 Torchvision the machine-vision package of torch. *MM '10 Proceedings of the international conference on Multimedia, New York*, 1485–1488, DOI: http://dx.doi.org/10.1145/1873951.1874254
4. **Oliphant T E** 2007 Python for scientific computing. *Computing in Science and Engineering* 9(3): 10–20. DOI: http://dx.doi.org/10.1109/MCSE.2007.58
5. **Bay H, Ess A, Tuytelaars T** and **van Gool L** 2008 Speeded-up robust features (surf). *Computer Vision and Image Understanding (CVIU)* 110(3): 346–359, DOI: http://dx.doi.org/10.1016/j.cviu.2007.09.014
6. **Haralick R M, Dinstein I** and **Shanmugam K** 1973 Textural features for image classification. *IEEE Transactions On Systems Man And Cybernetics* 3(6): 610–621, DOI: http://dx.doi.org/10.1109/TSMC.1973.4309314
7. **Ojala T, Pietikinen M** and **Menp T** 2002 Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(7): 971–987, DOI: http://dx.doi.org/10.1109/TPAMI.2002.1017623
8. **Hamilton N A, Pantelic N A, Hanson K** and **Teasdale R D** 2007 Fast automated cell phenotype image classification. *BMC bioinformatics* 8(110) DOI: http://dx.doi.org/10.1186/1471-2105-8-110
9. **Coelho L P, Ahmed A, Arnold A, Kangas J, Sheikh A-S, Xing E P, Cohen W W** and **Murphy R F** 2010 Structured Literature Image Finder: Extracting Information from Text and Images in Biomedical Literature. *Lecture notes in computer science* 6004: 23–32, DOI: http://dx.doi.org/10.1007/978-3-642-13131-8_4
10. **Daubechies I** 1990 The wavelet transform, time-frequency localization and signal analysis. *Information Theory, IEEE Transactions on* 36(5): 961–1005, DOI: http://dx.doi.org/10.1109/18.57199
11. **Felzenszwalb P** and **Huttenlocher D** 2004 Distance transforms of sampled functions. Technical report, Cornell University.
12. **Hunter J D** 2007 Matplotlib: A 2d graphics environment. *Computing in Science and Engineering* 9(3): 90–95, DOI: http://dx.doi.org/10.1109/MCSE.2007.55
13. **Dougherty E R** and **Lotufo R A** 2003 *Hands-on Morphological Image Processing.* Bellingham, WA: SPIE Press.
14. **Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn D** and **Smith K** 2011 Cython: The best of both worlds. *Computing in Science Engineering* 13(2): 31–39. DOI: http://dx.doi.org/10.1109/MCSE.2010.118
15. **Coelho L P, Shariff A** and **Murphy R F** 2009 Nuclear segmentation in microscope cell images: A hand-segmented dataset and comparison of algorithms. *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, 518–521, DOI: http://dx.doi.org/10.1109/ISBI.2009.5193098
16. **Coelho L P, Peng T** and **Murphy R F** 2010a Quantifying the distribution of probes between subcellular locations using unsupervised pattern unmixing. *Bioinformatics* 26(12): i7–i12. DOI: http://dx.doi.org/10.1109/ISBI.2009.5193098
17. **Cho B H, Cao-Berg I, Bakal J A** and **Murphy R F** 2012 Omero.searcher: content-based image search for microscope images. *Nature Methods* 9: 633–634, DOI: http://dx.doi.org/10.1038/nmeth.2086
18. **Mashburn D N, Lynch H E, Ma X** and **Huston M S** 2012 Enabling user-guided segmentation and tracking of surface-labeled cells in time-lapse image sets of living tissues. *Cytometry Part A* 81A(5): 409–418, DOI: http://dx.doi.org/10.1002/cyto.a.22034
19. **Machlek T** and **Oleviov K** 2013 Decentralized multi-agent algorithm for translational 2d image alignment. *Advances in Intelligent Systems and Computing* 183: 15–24, DOI: http://dx.doi.org/10.1007/978-3-642-32335-5_2
20. **Ploshnik E, Langner K M, Halevi A, Ben-Lulu M, Mller A H E, Fraaije J G E M, Agur Sevink G J** and **Shenhar R** 2013 Hierarchical structuring in block copolymer nanocomposites through two phase-separation processes operating on different time scales. *Advanced Functional Materials,* DOI: http://dx.doi.org/10.1002/adfm.201300091
21. **Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M** and **Duchesnay E** 2011 Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research* 12: 2825–2830.
22. **Demar J, Zupan B, Leban G** and **Curk T** 2004 Orange: From experimental machine learning to interactive data mining. *Lecture Notes in Computer Science* 3202: 537–539, DOI: http://dx.doi.org/10.1007/978-3-540-30116-5_58
23. **Schaul T, Bayer J, Wierstra D, Sun Y, Felder M, Sehnke F, Rückstieß T** and **Schmidhuber J** 2010 Pybrain. *The Journal of Machine Learning Research* 11: 743–746.
24. **Sonnenburg S, Rätsch G, Henschel S, Widmer C, Behr J, Zien A, Bona F, Binder A, Gehl C** and **Franc V** 2010 The shogun machine learning toolbox. *The Journal of Machine Learning Research* 11: 1799–1802.
25. **Beazley D** 2003 Automated scientific software scripting with swig. *Future Generation Computer Systems - Tools for Program Development and Analysis* 19(5): 599–609. DOI: http://dx.doi.org/10.1016/S0167-739X(02)00171-1.
26. **Zhao T** and **Murphy R F** 2006 Automated interpretation of subcellular location patterns from three-dimensional confocal microscopy. In: *Handbook Of Biological Confocal Microscopy.* Springer US, pp. 818–828, DOI: http://dx.doi.org/10.1007/978-0-387-45524-2_47
27. **Evans C** 2009 Notes on the OpenSURF Library SURF: Speeded Up Robust Features. Available at: https://nll.googlecode.com/svn-history/r1367/trunk/references/opensurf.pdf [accessed 24 July 2013].

How to cite this article: Coelho, L P 2013 Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software* 1:e3, DOI: http://dx.doi.org/10.5334/jors.ac

Published: 26 July 2013