O'Mara, A, et al. 2017 ImageSURF: An ImageJ Plugin for Batch Pixel-Based Image Segmentation Using Random Forests. *Journal of Open Research Software,* 5: 31. DOI: https://doi.org/10.5334/jors.172

SOFTWARE METAPAPER

ImageSURF: An ImageJ Plugin for Batch Pixel-Based Image Segmentation Using Random Forests

Aidan O'Mara[1], Anna E. King[1], James C. Vickers[1] and Matthew T. K. Kirkcaldie[2]
[1] Wicking Dementia Research and Education Centre, Faculty of Health, University of Tasmania, AU
[2] School of Medicine, Faculty of Health, University of Tasmania, AU
Corresponding author: Aidan O'Mara, PhD (omaraa@utas.edu.au)

Image segmentation is a necessary step in automated quantitative imaging. ImageSURF is a macro-compatible ImageJ2/FIJI plugin for pixel-based image segmentation that considers a range of image derivatives to train pixel classifiers which are then applied to image sets of any size to produce segmentations without bias in a consistent, transparent and reproducible manner. The plugin is available from ImageJ update site http://sites.imagej.net/ImageSURF/ and source code from https://github.com/omaraa/ImageSURF.

Keywords: ImageJ; FIJI; segmentation; trainable segmentation; binary segmentation; random forests
Funding statement: This research was supported by an Australian Government Research Training Program Scholarship.

(1) Overview

Introduction

A critical task in quantitative imaging is segmentation, in which pixels are partitioned into distinct groups. For example, fluorescent labelling in microscopic images might be segmented as signal and background, or photographs of botanical specimens might be segmented as specimen and background (**Figure 1**). After segmentation, measurements and analyses may be performed to determine, for example, the size, shape, coverage, spatial distribution or morphology of the identified features.

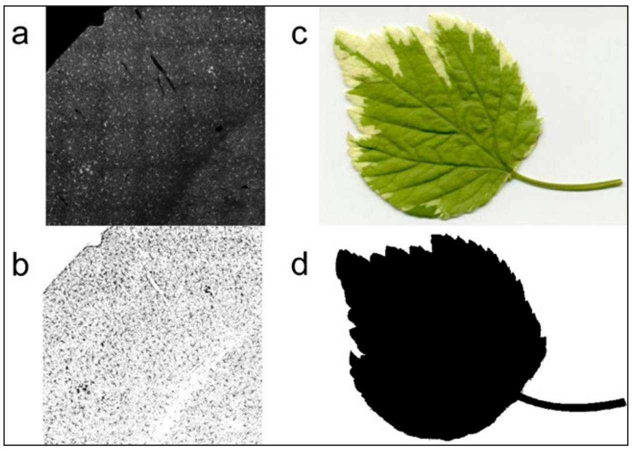


Figure 1: Confocal fluorescence image of Iba-1 labelled microglial cells in APPswe/PS1dE9 mouse brain tissue **(a)** and segmented image **(b)**. ImageJ sample image leaf.jpeg **(c)** and segmented image **(d)**.

Many segmentation techniques such as thresholding, where pixels above a selected intensity threshold are discriminated from background [1, 2], depend exclusively on the brightness of individual pixels, making them sensitive to noise and regional variations in intensity [1]. Parameters or seeds for segmentation are often manually selected on a per-image basis based on a preview of the result, or may be selected, reviewed and refined in an unstructured iterative process [3].

The use of image segmentation in research raises several reproducibility issues. In manual segmentation, parameter choice is influenced by conditions such as screen brightness and dynamic range, ambient light, perceived brightness, and subjective bias [4] especially in unblinded raters [5], and such factors are rarely reported, limiting reproducibility. When automated segmentation tools are used, they are often commercial platforms whose detailed algorithms are proprietary. This complicates comparisons between studies and poses replication problems, as legacy software and hardware may no longer be available, and algorithms or interfaces may differ between versions.

Open-source trainable segmentation tools, such as Ilastik [3] or the Trainable Segmentation [6] plugin for ImageJ [7], address many of these issues by using supervised machine learning algorithms to study 'training set' of pixels, which have been manually assigned class annotations, and create a model ('classifier') to reliably discriminate between these classes. The context of each pixel (e.g., intensity, texture, edges, entropy) can be considered, making the classifier more robust to image artifacts and intensity shifts [3]. After training, a classifier can be saved and used to perform objective and repeatable segmentation of large numbers of similarly processed images. Although Ilastik and Trainable Segmentation have limited support for importing and exporting image annotations, neither supports both batch import and batch export of these annotations in standard formats. These limitations make it difficult to reproduce and update classifiers, to use alternative software and input devices for annotation, and to share training sets in a transparent and collaborative manner.

For biomedical microscopy, the advantage of machine learning image segmentation is the ability to apply a single classifier to large image sets which vary somewhat in brightness, background level and other image attributes. For our particular application, high resolution multichannel confocal fluorescence images of rodent brains were routinely 22,000 × 18,000 pixels or larger, and 5 or more images per animal need to be segmented to produce useful quantitative data. In order to train a classifier that is robust to the variation across these large image sets it is beneficial to create a training set consisting of smaller cropped images of randomly selected regions in these images. To train a classifier with such large sets of input images (potentially hundreds) it is necessary to use an offline iterative process of class annotation, training and validation, rather than the 'live preview' workflows offered by Ilastik and Trainable Segmentation. To cope with offline iterative training on the sets of large images generated by confocal and fluorescence microscopy, and the very large amount of computed pixel feature information, data structures and algorithms with reduced processing and storage requirements are needed.

In this context, we determined that Trainable Segmentation was unable to use large training sets of arbitrarily sized annotated images without an extensive re-write due to the size and complexity of WEKA Instance data structures. Using a custom implementation of WEKA Instance backed by primitive arrays slightly reduced memory usage, but also increased computation time. Trainable Segmentation only allows import and export of training data with calculated image features in the text-based WEKA arff file format which is wasteful of storage space and slow to import/export for large images, making iterative annotation of large image sets unworkably slow. Furthermore, Trainable Segmentation is a legacy ImageJ1 plugin, limiting its interoperability with SciJava and ImageJ2-compatible applications such as OMERO, KNIME and MiToBo [7].

We therefore developed Image Segmentation Using Random Forests (ImageSURF), a freely-available open-source pixel-classification plugin for ImageJ2/FIJI [7] to meet our requirements. ImageSURF uses standard bitmap formats for class annotations, making the training process open, repeatable and able to incorporate large training sets created by multiple users across multiple sessions with the software of their choice. ImageSURF uses primitive data structures to avoid the substantial overheads of Object data structures such as the WEKA Instance.

We are currently using ImageSURF to study the aggregation and deposition of amyloid-β peptide in brain tissue of Alzheimer's disease rodent models by means of immunolabelling and confocal microscopy. Once trained, ImageSURF is a drop-in replacement for threshold segmentation in our ImageJ scripts.

Implementation and architecture

ImageSURF is an ImageJ2/FIJI plugin written and compiled using Java 1.8, with the user interface classes implementing the SciJava Command interface.

Training input is read from three corresponding sets of images – a set of raw single-plane single- or multi-channel greyscale images, a set of images in RGB format that have been intensity-scaled and pseudo coloured as appropriate for manual annotation, and a set of these RGB images with class annotations in distinct colours. The class annotations are read by taking the difference of the unannotated and annotated RGB images. Each distinct annotation colour is assigned a class index based on the hexadecimal RGB value. ImageSURF supports up to 128 classes.

Annotation images are manually created in ImageJ using the paint tools or using bitmap image software such as Adobe Photoshop or GIMP on any device such as a desktop with a drawing tablet input or portable touchscreen device (**Figure 2**).

ImageSURF classifiers are built from the training input using an optimised implementation of the random forests algorithm [8] adapted from the FastRandomForest [9] plugin for the WEKA environment [10], which is the default classifier used by Trainable Segmentation [6, 7]. Features are stored as size-efficient primitive

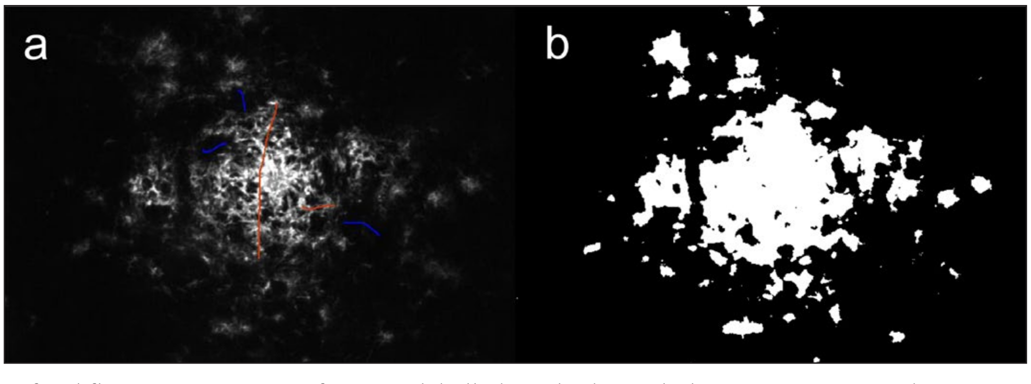


Figure 2: Confocal fluorescence image of MOAB2 labelled amyloid-β pathology in APPswe/PS1dE9 mouse brain tissue with sparse annotations for signal (red) and background (blue) **(a)** and resulting segmentation **(b)**.

data structures (byte or short arrays for 8-bit and 16-bit images, respectively) and may be pre-calculated and saved to disk. These optimisations reduce flexibility and functionality compared to Trainable Segmentation by making ImageSURF incompatible with the wide range of WEKA classifiers and analysis tools, but substantially increases its capacity for working with large training sets and images while maintaining compatibility with ImageJ workflows, including pre- and post-processing tools and analysis pipelines.

Pixel features are calculated using filters across circular neighbourhoods with various radii (**Figure 3**). In the current release of ImageSURF we have implemented a filter set to suit confocal fluorescence images of amyloid-β, including mean, minimum, maximum, median, Gaussian, standard deviation, range, difference of Gaussians, difference from mean, minimum, maximum, median and Gaussian, locally scaled intensity, entropy and difference of entropy. Radii are selected as a series of values within the integer set k = $2^a + 1$ (3, 5, 9, 17, 33, 65, 129, 257…). For further efficiency, a filter dependency tree is used to reduce repeat operations: e.g., the output of Gaussian radii *r* and *s* is reused for the difference-of-Gaussians with those radii.

Pixel features can be pre-calculated and cached to substantially reduce computation when re-training classifiers on modified or extended training sets, or as part of an image analysis pipeline where feature images are automatically calculated immediately after image acquisition to speed up later training. Pixel features that can be derived from other saved features with minimal processing are not cached in order to reduce disk usage and read-times.

ImageSURF supports multi-channel images by calculating pixel features for each channel, and all combinations of channels merged in grayscale, by averaging. E.g., for a three channel RGB image, each image filter would be applied to the red, green and blue channels, combined red/green, red/blue, green/blue and red/green/blue to produce seven sets of pixel features. This allows ImageSURF to consider information from all channels and the interactions between channels.

After a classifier has been trained, a subset of the most important features is selected using a modified version of Breiman's feature importance calculation algorithm [8] as implemented in Supek's FastRandomForest [10]. For each feature, the classifier is applied to the training set with the values for that feature randomly shuffled. If classification accuracy remains high, that feature is less important and is ranked accordingly. After feature selection the classifier is re-trained considering only the most important features. This optimisation substantially reduces the computation and disk space required when pre-calculating features. Using a minimal set of image features also reduces the memory requirements for image segmentation. The parameters used to train a classifier, including the image features applied to images, can be viewed using the ImageSURF Get Classifier Details command.

ImageSURF also supports segmentation of multi-dimensional images on a plane-by-plane basis. Each

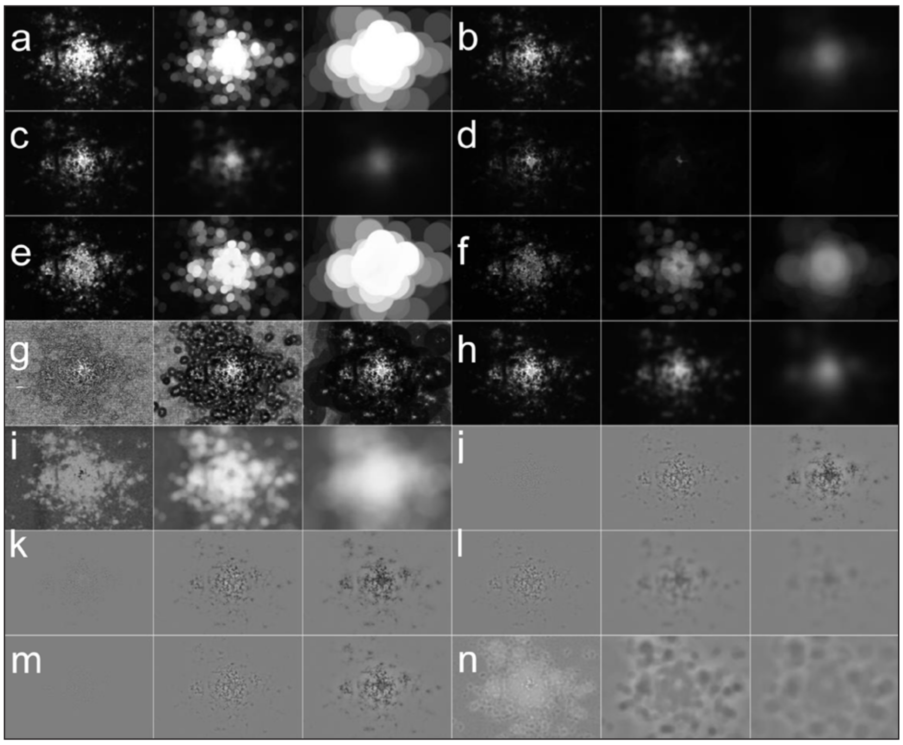


Figure 3: Confocal fluorescence image of MOAB2 labelled amyloid-β pathology in APPswe/PS1dE9 mouse brain tissue with image filters applied. Maximum **(a)**, mean **(b)**, median **(c)**, minimum **(d)**, range **(e)**, standard deviation **(f)**, locally scaled intensity **(g)**, Gaussian **(h)**, entropy **(i)**, difference from mean **(j)**, difference from median **(k)** and difference from Gaussian **(l)** filters at radii 3, 17 and 65 pixels. Difference-of-Gaussians at 3/17, 17/65, and 33/65 pixels **(j)**. Difference-of-entropy for 3/17, 17/65, and 33/65 pixels **(k)**.

two-dimensional image plane is segmented independently to produce an output image stack with the same dimensions as the input image stack.

Quality control

Automated testing of pixel-based segmentation tools that use sparsely annotated training images is non-trivial, particularly when evaluation of the output requires subjective judgement that is not captured by the annotations. Therefore, we provide a small set of example images and class annotation training sets, along with instructions for end-to-end testing and usage of ImageSURF.

All sample images and label files are contained in the ImageSURF MOAB2 images example [11]. ImageSURF commands are in the Plugins >> Segmentation >> ImageSURF menu in ImageJ2 and FIJI (**Figure 4**):

1) Configure the classifier settings using the ImageSURF Classifier Settings command. Use the default ImageSURF settings (**Figure 5**).
2) Select the image filters using the Select ImageSURF Features command. Exclude the entropy and median filters and their derivatives as these may take some time to calculate. Set the filter radius range as 0–33 (**Figure 6**).
3) Train the classifier using the Train ImageSURF Classifier command. Set the raw, un-annotated and annotated image paths. Set an appropriate classifier output path and select "Segment training images and display as stacks" to verify the classifier accuracy after training (**Figure 7**) The training and segmentation process takes approximately 5 minutes on a modern quad-core computer. Detailed progress information is displayed in the ImageJ console.

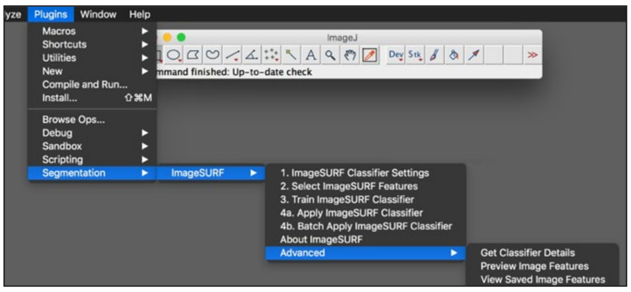
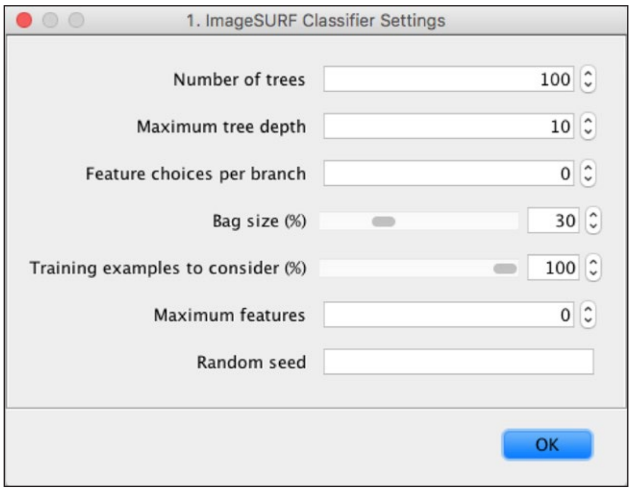
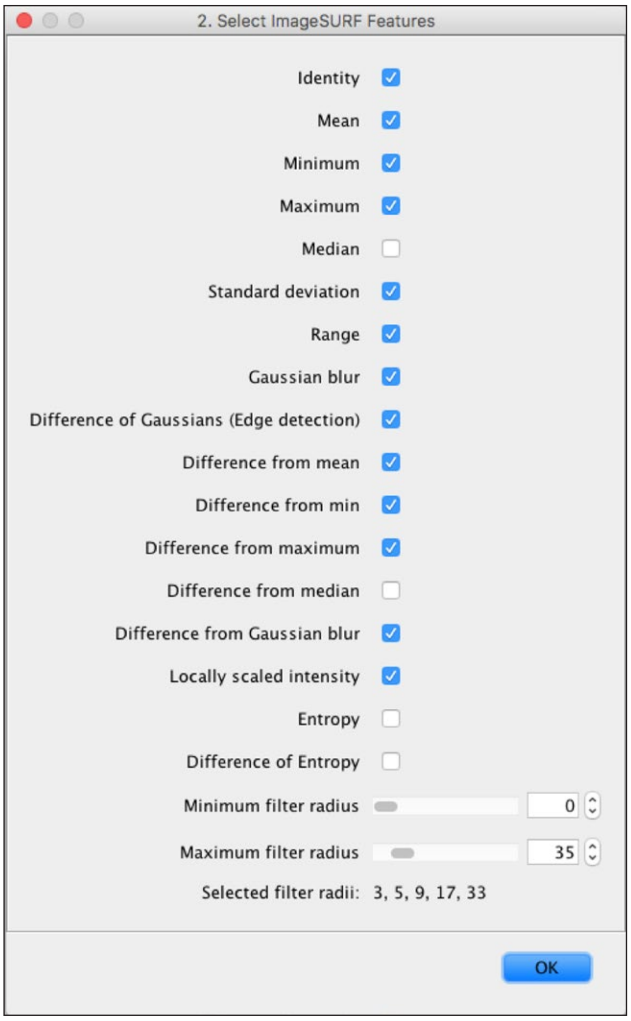


Figure 4: ImageSURF menu options.


1. ImageSURF Classifier Settings

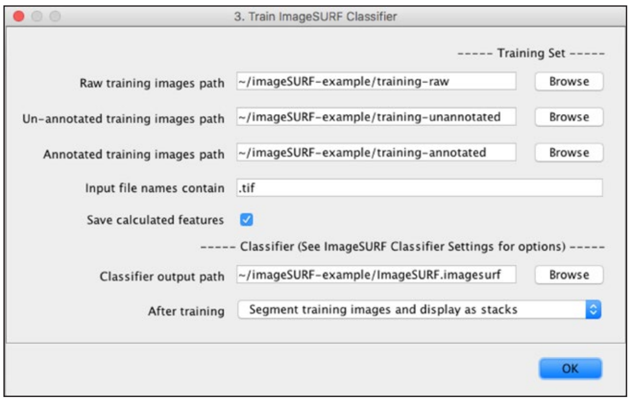
Number of trees	100
Maximum tree depth	10
Feature choices per branch	0
Bag size (%)	30
Training examples to consider (%)	100
Maximum features	0
Random seed	

Figure 5: ImageSURF classifier settings dialog with default options.


2. Select ImageSURF Features

Identity	✓
Mean	✓
Minimum	✓
Maximum	✓
Median	
Standard deviation	✓
Range	✓
Gaussian blur	✓
Difference of Gaussians (Edge detection)	✓
Difference from mean	✓
Difference from min	✓
Difference from maximum	✓
Difference from median	
Difference from Gaussian blur	✓
Locally scaled intensity	✓
Entropy	
Difference of Entropy	
Minimum filter radius	0
Maximum filter radius	35
Selected filter radii:	3, 5, 9, 17, 33

Figure 6: ImageSURF filter selection dialog with example filters selected.


3. Train ImageSURF Classifier

----- Training Set -----

Raw training images path	~/imageSURF-example/training-raw
Un-annotated training images path	~/imageSURF-example/training-unannotated
Annotated training images path	~/imageSURF-example/training-annotated
Input file names contain	.tif
Save calculated features	✓

----- Classifier (See ImageSURF Classifier Settings for options) -----

Classifier output path	~/imageSURF-example/ImageSURF.imagesurf
After training	Segment training images and display as stacks

Figure 7: ImageSURF classifier training dialog with example settings.

4) Verify the accuracy of the trained classifier applied to the training images (**Figure 8**). Segmentations may be overlaid using built-in ImageJ tools, e.g., Montage and Merge Channel. If any problem areas are identified, more examples can be added to the training set and the classifier re-trained.
5) Segment the full dataset using the saved classifier. A classifier can be used to segment an open image using the Apply ImageSURF Classifier command or to a folder of images using Batch Apply ImageSURF Classifier.

We recommend using an iterative process of annotation and verification to train ImageSURF classifiers as shown in **Figure 9**.

(2) Availability

Operating system

ImageSURF is compatible with ImageJ2 software running on a Java Virtual Machine version 8 or above. As of writing ImageJ2 is available for macOS, Linux and Windows operating systems.

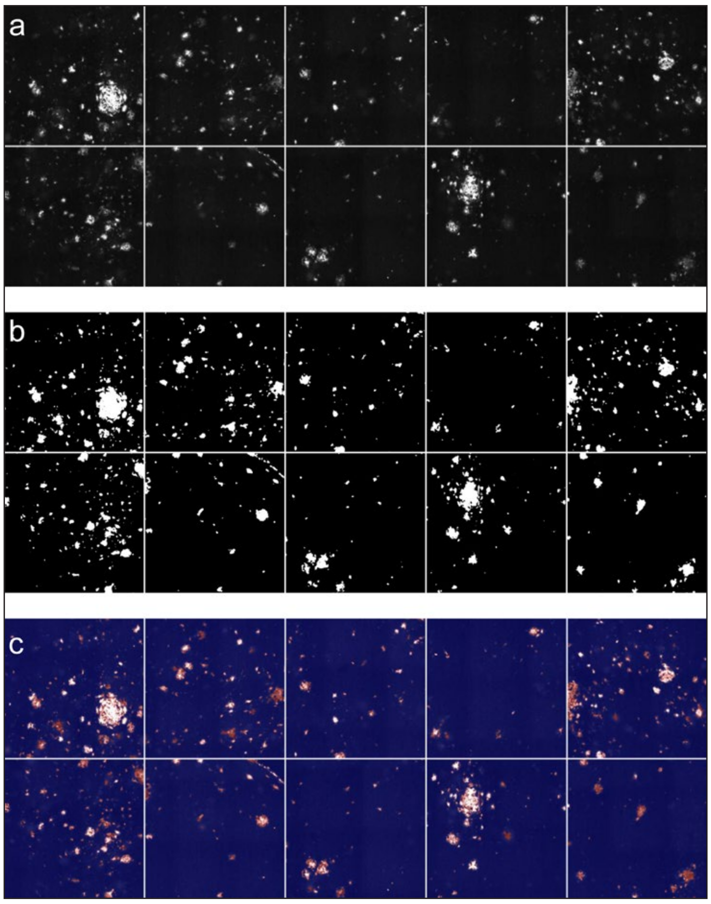


Figure 8: ImageSURF training examples. Confocal fluorescence images of MOAB2 labelled amyloid-β pathology in APPswe/PS1dE9 mouse brain tissue **(a)**. Segmented training images **(b)** and merged image **(c)** using the ImageJ Merge Channels tool to display the segmented signal pixels as transparent red and background as transparent blue.

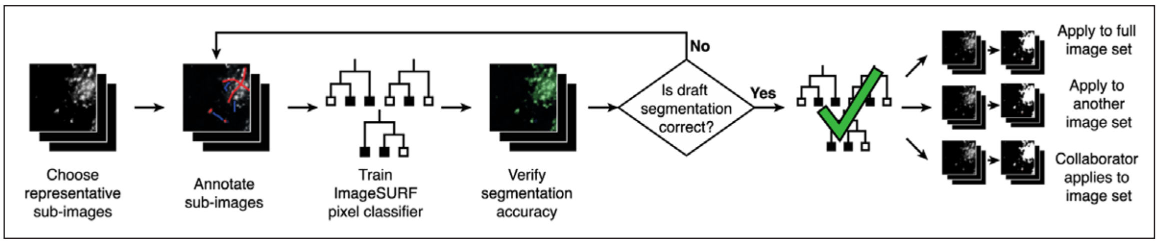

Choose representative sub-images
Annotate sub-images
Train ImageSURF pixel classifier
Verify segmentation accuracy
Is draft segmentation correct?
No
Yes
Apply to full image set
Apply to another image set
Collaborator applies to image set

Figure 9: ImageSURF pixel classifier training workflow. A representative set of sub-images are selected and cropped from the full image set and sparsely annotated as signal or background using a bitmap image software package. The sub-images and annotations are used as the input to train an ImageSURF classifier which is them applied back to the input sub-images. The accuracy of the sub-image segmentations is manually verified and the annotation training and verification processes repeated until the sub-image segmentation is accurate. Once the trained classifier has been verified as accurate, it can be applied to any image set of which the training set is representative.

Programming language
ImageSURF is written in Java 1.8.

Additional system requirements
No further requirements. 8 GB of RAM or greater is recommended.

Dependencies
ImageJ2 (tested with ImageJ 2.0.0-rc-61).

Java Primitive 1.2.3 or greater (https://github.com/mintern-java/primitive) must be loaded by ImageJ. It will be loaded automatically if the release .jar file is included in the ImageJ2 plugins directory or if ImageSURF was installed using the ImageJ2 updater.

List of contributors
Aidan R O'Mara (programming, testing and validation).
Matthew Kirkcaldie (testing and validation).

Software location
Archive Zenodo
Name: ImageSURF
Persistent identifier: https://doi.org/10.5281/zenodo.819556
Licence: GNU GPL version 3
Publisher: Aidan R O'Mara
Version published: v1.1.1
Date published: 27/06/17

Code repository GitHub
Name: ImageSURF
Identifier: https://github.com/omaraa/ImageSURF
Licence: GNU GPL version 3
Date published: 27/06/17

Language
English

(3) Reuse potential
ImageSURF is a robust and general-purpose segmentation tool that is not limited to any particular field or application. ImageSURF can be used as a drop-in replacement for other binary segmentation tools in ImageJ scripts and as a SciJava command in compatible image processing pipelines.

ImageSURF can be extended to capture a wider range of image features by implementing more image filters. It may be necessary to add support for mixed value types (e.g., a FeatureReader class that supports both unsigned short and 32-bit floating point image features) for image filters that produce output that cannot be scaled to an unsigned byte or short value without substantial loss of information.

To practically implement support for true generic n-dimensional segmentation using n-dimensional filters would entail losing the benefits of using primitive data structures, due to current limitations of the Java language and virtual machine. This may be re-visited when Java includes support for specialised generics and value types as discussed in the OpenJDK mailing lists and events [12]. Support for 3-dimensional segmentation could be added by implementing a second, 3-dimensional version of each image filter.

Support for modifying and using ImageSURF is available through the GitHub issues page (https://github.com/omaraa/ImageSURF/issues) and GitHub wiki (https://github.com/omaraa/ImageSURF/wiki).

Acknowledgements
The authors thank Robert Ollington and Carolyn King for their contributions to the initial stages of this project.

Competing Interests
The authors have no competing interests to declare.

References
1. **Pham, D L, Xu, C** and **Prince, J L** 2000 Current methods in medical image segmentation. *Annual review of biomedical engineering*, 2: 315–337. DOI: https://doi.org/10.1146/annurev.bioeng.2.1.315
2. **Roeder, A H K, Cunha, A, Burl, M C** and **Meyerowitz, E M** 2012 A computational image analysis glossary for biologists. *Development*, 139(17): 3071–3080. DOI: https://doi.org/10.1242/dev.076414
3. **Sommer, C, Straehle, C, Kothe, U** and **Hamprecht, F A** 2011 Ilastik: Interactive learning and segmentation toolkit. *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on*, 230–233. DOI: https://doi.org/10.1109/ISBI.2011.5872394
4. **Teverovskiy, M, Vengrenyuk, Y, Tabesh, A, Sapir, M, Fogarasi, S, Pang, H-Y, Khan, F M, Hamann, S, Capodieci, P** and **Clayton, M** 2008 Automated localization and quantification of protein multiplexes via multispectral fluorescence imaging. *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*, 300–303. DOI: https://doi.org/10.1109/ISBI.2008.4540992
5. **Jucker, M** 2010 The benefits and limitations of animal models for translational research in neurodegenerative diseases. *Nature medicine*, 16(11): 1210–1214. DOI: https://doi.org/10.1038/nm.2224
6. **Arganda-Carreras, I, Kaynig, V, Rueden, C T, Schindelin, J, Cardona, A** and **Seung, H S** 2016 Trainable_Segmentation: Release v3.1.2. DOI: https://doi.org/10.5281/ZENODO.59290
7. **Schindelin, J, Rueden, C T, Hiner, M C** and **Eliceiri, K W** 2015 The ImageJ Ecosystem: An Open Platform for Biomedical Image Analysis. *Molecular Reproduction and Development*, 82: 518–529. DOI: https://doi.org/10.1002/mrd.22489
8. **Breiman, L** 2001 Random forests. *Machine learning*, 45(1): 5–32. DOI: https://doi.org/10.1023/A:1010933404324
9. **Frank, E, Hall, M A** and **Witten, I H** 2016 The WEKA Workbench. *Morgan Kaufmann*, Fourth Edition.
10. **Supek, F** 2015 FastRandomForest. *Google Code*. Available at: https://code.google.com/archive/p/fast-random-forest/.

11. **O’Mara, A R, Collins, J M, King, A E, Vickers, J C** and **Kirkcaldie, M T K** 2017 ImageSURF MOAB2 Image Examples. DOI: https://doi.org/10.5281/zenodo.819511

12. **Rose, J** and **Goetz, B** 2017 Minimal Value Types. Retrieved from: http://cr.openjdk.java.net/~jrose/values/shady-values.html May 2017.

How to cite this article: O’Mara, A, King, A E, Vickers, J C and Kirkcaldie, M T K 2017 ImageSURF: An ImageJ Plugin for Batch Pixel-Based Image Segmentation Using Random Forests. *Journal of Open Research Software,* 5: 31. DOI: https://doi.org/10.5334/jors.172

Submitted: 28 March 2017 **Accepted:** 29 September 2017 **Published:** 06 November 2017