Title:

Trainable image segmentation combining deep image features with random forest classification.

Authors/Affiliations:

Christian Tischer\* (1,3)

Ignacio Arganda-Carreras (?)

Yannick Schwab (4)

Rainer Pepperkok (3)

1) Centre for Bioimage Analysis, European Molecular Biology Laboratory, Meyerhofstrasse 1, 69117 Heidelberg, Germany.

2) Structural and Computational Biology Unit, European Molecular Biology Laboratory, Meyerhofstrasse 1, 69117 Heidelberg, Germany.

3) Advanced Light Microscopy Core Facility, European Molecular Biology Laboratory, Meyerhofstrasse 1, 69117 Heidelberg, Germany.

4) Electron Microscopy Core Facility, European Molecular Biology Laboratory, Meyerhofstrasse 1, 69117 Heidelberg, Germany.

\*Corresponding author

Contact: Christian.Tischer@EMBL.DE

Abstract

Analysis of scientific images typically requires the segmentation of structures of interest. As manual image segmentation is very tedious and time consuming machine learning approaches have been developed to automate this task. Initial developments were based on extracting image features and, based on sparse user annotations, train a random forest classifier (RF) to predict the segmentation of unlabeled image regions. Recently, using deep convolutional image features implemented in a neural network architecture have been shown to outperform the initial RF based approaches. However, building NN based interactive tools is challenging as current algorithms for learning the NN connection weights from training data typically take several hours. Here, we present a novel approach that uses deep convolutional image features as an input to a RF classifier, thereby creating an interactive tool combining the power of deep convolution with the convenience of RF classification.

Introduction

Analysis of scientific image data typically requires partitioning of images into objects such as for instance cell nuclei and intracellular organelles in biological research. It is in principle possible to perform such image segmentation manually. However, the amount of data produced by modern imaging modalities such as high-throughput, light-sheet and focused-ion-beam scanning electron microscopy is on the order of Giga- to Tera-bytes, rendering manual segmentation almost impossible. Thus, automated image segmentation approaches are of great interest. In fluorescence microscopy (FM) automated image segmentation can sometimes be achieved using simple intensity thresholding that can be readily performed using freely available software packages such as ImageJ or CellProfiler [Ref\_IJ, Ref\_CP]. However, already distinguishing tubular from dot-like structures in FM images requires more sophisticated image processing procedures that might not be known to the broader scientific community. Automated segmentation of electron microscopy (EM) images typically poses an even bigger challenge as there can be many different structures with albeit very similar gray level distributions. In order to segment such images automatically it is typically necessary to combine tens to hundreds of various local image features, which, e.g., compute local intensity edges, ridges, or more complex textures. Even for an image analyst expert, finding the right combination of image features for segmenting different objects can be extremely time consuming. However, there exist user friendly machine learning solutions such as ilastik [Ref\_Ilastik] and ImageJ’s Trainable Weka Segmentation Plugin [Ref\_IJTWS]. In both tools several predefined image features are computed automatically and, based on sparse human annotations, a random forest (RF) classifier [Ref\_Breiman] is trained to combine those feature values to assign each image pixel to a different structure (class). While this approach is very powerful and has proven useful in several occasions (@All: Examples?) deep convolutional neural networks (DCNN) seem even more promising in terms of image segmentation capabilities as they have achieved high ranking results in segmentation challenges [Ref\_UNet, Ref\_3DUNet]. However, one disadvantage of DCNNs is that a full training of the neural network using current algorithms and graphic cards still takes hours to days [Ref\_3DUNet]. This is in stark contrast to RF classifiers which can be trained in seconds to minutes. Current user friendly approaches thus consist of partly pre-trained NNs in order to keep training times at bay [Ref\_ZEISS, Ref\_American\_Company]

In this article we describe an alternative approach combining the power of deep convolution with the convenience of training of a random forest classifier. Our tool is freely available as a user friendly and big image data compatible Fiji plugin [Ref\_FijiPlugin].

Implementation overview

Our trainable deep feature random forest (DFRF) image segmentation is implemented in Java as a Fiji plugin and can be readily installed via Fiji's update manager [Ref\_FijiPlugin]. The graphical user interface (GUI) and some of the underlying code is based on the Trainable Weka Segmentation (TWS) ImageJ plugin [Ref\_TWS]. In the GUI, the user can configure the number of classes to be trained and draw annotations on the image (see Fig\_GUI). During training, a feature vector is computed for each annotated pixel and used to train a random forest (RF) classifier [Ref\_WEKA]. Once a classifier has been trained it can be applied either to the whole data set or a configurable region of interest (ROI). Classification results are shown as a color overlay on the input data and can be exported in various formats.

Feature computation

The main difference of our work to ilastik and TWS is a deep feature computation scheme, which is depicted in figure Fig\_Scheme. From the input image we compute eigenvalues of the structure tensor (ST) and the hessian matrix (HM), where in our implementation both ST and HM accommodate spatially anisotropic input data [Ref\_ImageScience]. Choosing solely ST and HM eigenvalues as features was motivated three-fold. ST and HM are based on 1st and 2nd image derivatives, which - together with the input image (0th derivative) – resemble the beginning of a Taylor expansion and should thus have a good descriptive power [Ref\_TaylorExpansion]. Moreover, ST and HM eigenvalues are rotationally invariant, making them particularly suited for biological data which typically had no distinct orientation relative to the coordinate system of the image acquisition device. Finally, HM eigenvalues have proven as powerful descriptors of vesicular, tubular and membranous structures [Ref\_Missing] and seem thus perfectly adapted to biological image data, which is our main application.

Next - similar to conventional DCNN architectures - raw, ST and HM images are binned and again subjected to ST and HM eigenvalue computation (see Fig\_Scheme). This process is repeated N times, where N and bin widths at each step can be adapted to accommodate the spatial length scales of the respective input image (see Suppl. Mat. for details). As in DCNN, this procedure yields a collection of images of different sizes, which we up-sample accordingly such that all images have the size as the input image. Naturally, up-sampled images from deep layers will have a smooth appearance as they contain information about long length-scales. Finally, as in ilastik or TWS, for each annotated pixel we compile a feature vector and train a conventional RF classifier.

Big image data compatibility

For both input and output data the user has the choice to either keep them fully in RAM (for small data sets) or load and write portions from and to disk on demand (for large data sets) [Ref\_BigDataTools]. In addition, our implementation performs classification using an automated tiling strategy such that also here is no limitation in terms of the data set size with respect to the available RAM. Finally, for users having access to a (Slurm) cluster, there is the option to automatically process the data in a distributed fashion as multiple jobs.

Results

Transfer learning:

* Quite good transfer learning when using CLAHE.
  + Examples: Apply 6.3min trained data onto 5.3min

Discussion

Our general DCRF scheme is compatible with any kind of image features. One could readily add more features or make the features user selectable. However, we opted for a fixed and rather small feature set of raw data, ST and HM eigenvalues for multiple reasons. Firstly, it is in practice confusing having too much choice at the feature selection step, especially for non-expert users, which are not familiar with, e.g. ST and HM eigenvalues. Secondly, using more features leads to an exponential explosion of the number of features in any deep convolutional computation scheme.

We decided to keep convolution depth N and down-sampling scheme flexible in order to allow the user to adapt the feature computation scheme to the different spatial lengths scales in the different input data.

Following our scheme for a 3-D input image the number of images at depth N is 7^N. In order to be able to go deep without an explosion of the number of feature images, we introduced a maximal feature depth (MFD) parameter. With this parameter one can control how many "features of features" are allowed. For example, choosing a value of two would only allow features of the following structure: Orig\_HME\_Bin\_Bin\_STE or Orig\_STE\_Bin\_HME but not Orig\_HME\_Bin\_STE\_Bin\_STE. Like this the number of features at depth N is (@TODO: figure out the formula).

### Uncertainty display and navigation

As it for instance is possible in ilastik [Ref\_Ilastik] one can activate an uncertainty overlay, showing the classification margin, i.e. the difference between the most and second most likely class probabilities. This helps the user to see where more labelling is needed. However, while this is very useful, we found that is is inefficient to manually find and visit regions of high uncertainty in large data sets. We thus implemented an "uncertainty navigation". During classification we keep track of the average uncertainty in each classified image block and store this information in a sorted list. Using keyboard shortcuts the user can navigate through this list and the corresponding region is highlighted on the input image, enabling the user to efficietnly add more class labels in regions of high uncertainty.

### N-D support

The DFRF plugin supports multi-channel and multi-time-point data. In terms of multi-channel support the user can choose which channels should be taken into account for the feature computation. Fatures are computed in all channels independenly; we currently do not compute features combining gray values from multiple channels.

### Storage of training data

The user annotations (labels) for each image can be stored and reloaded. The stored file contains the label ROIs, the respective class label and also, if they were computed already, the feature values.

## User guide

### Feature naming scheme

The names of the respective features indicate whether either the hessian matrix (He) or the structure tensor (St) was computed and which eigenvalue was computed, the largest (L), the middle (M), or the smallest (S). In addition the name contains information about the current binning relativ to the original image. For example, 9x9x3\_StM\_3x3x1\_StS\_Orig means that first the smallest eigenvalue of the structure tensor (StS) was computed followed by a 3x3x1 average binning, followed by computing the middle eigenvalue of the structure tensor (StM), followed by a 3x3x3 binnning yielding a 9x9x3 binned image with respect to the original image.

#### Down-sample factor

The down-sample factor (DSF) determines how much the images are down-sampled from one resolution level (L) to the next. The optimal choice depends on the spatial structure of the input data. In general, a smaller value captures more details while a lager value faster approaches larger spatial scales. For example choosing a downsample factor of 2 yields binnings 2^(L) (1,2,4,8,...) while 3 yields binnings of 3^(L) (1,3,9,27,...); for the data shown in this article we used a down-sample factor of 3.

#### Maximal feature depth

In our current architecture we have 6 features (4 for 2-D images), namely 3 (2) eigenvalues of the hessian matrix and 3 (2) eigenvalues of the structure tensor. In addition, we allow also simply keep the downsampled version of each image without an additional feature computed. Thus, at each resolution level there are 7 (5) images that could be derived from all images in the preceeding resolution layer, yielding for the 3-D case 7^(L+1) features (L0:7, L1:49, L2:343, L3:2401, L4:16807, ...) at downsampling level L. Obviously this is a fast growing number, which for instance slows down the RF training that has to test all those features for their usefulness given the current classification task. To keep the numbers of features at bay we thus introduced a maximal feature depth (MFD) which determines how many levels of "features of features" are allowed. For example, given a downsample factor of 3 a feature image at level 2 could be 9x9x9\_HeL\_3x3x3\_StS\_1x1x1\_Orig. Given a MCD of 2, this feature image could not be subjected to any further filters, but could only be further downsampled. However the image 9x9x9\_3x3x3\_StS\_1x1x1\_Orig would be subjected to additional filters, because it was so far only subjected to one filter, namely the smallest eigenvalue of the structure tensor (StS) at resolution level 0.

## Applications

- We present the segmentation of several challenging EM data sets, showing that our approach is

- interactive

- accurate

- fast

- big data compatible

### Conceptual example application: line of dots

Before tacking on actual data we demonstrate the concept of our strategy on a simple manufactured example, namely a 2-D image comprising several structures including a line of dots (Figure\_LineOfDots). Probably there are several ways of segmenting the line of dots using our frame-work, we simply show one that felt intuitive to us. The smallest eigenvalue of the structure tensor highlights dots, corners, and ends of lines (Orig\_StS), thereby already eliminating a number of bright pixels that are not part of the line of dots. Two times 3x3 average binning connects the dots (Orig\_StS\_3x3\_3x3). To distinguish pixels on the line from the remaining dots, we compute the largest eigenvalue of the hessian matrix (Orig\_StS\_3x3\_3x3\_HeL), which has large negative values only at dots. Finally, we upsample the last two images and apply a two node decision tree, namely Orig\_StS\_3x3\_3x3\_UpSample > 60 => Orig\_StS\_3x3\_3x3\_HeL\_UpSample > -15. As one can see, this recipe sucessfully solves this non-trivial segmentation problem.

### C.elegans FIB-SEM

- We have manual ground truth => we can give accuracies

### HeLa interphase cell FIB-SEM

### HeLa mitoic cells FIB-SEM

- measurement: fraction of DNA covered with ER

### ISBI data set

- Ignacio

## Discussion

- To go from one resolution layer to the next we opted for an average binning. Binning has the advantage that the image size decreases such that both computation times and memory requirements decrease as well. For example, for a 3-D input image and a downsampling factor of 3 the number of pixels is reduced by 3x3x3=27 at each resolution. This more than compensates the 7-fold increase in the number of images at each resolution such that computation times and memory requirements are in fact decreasing at deeper resolution layers. This is in stark contrast to the current implementations in ilastik and TWS where larger features are computed by increased kernel width, which does not decrease the memory requirements and, depending on implementation details, even increases the computation times.

- In neural network implementations of deep convolutions all convolutions are learned during the training. This has the advantage that the NN has the chance to learn the optimal convolutional filters for the given segmentation task. However this comes at a cost of many parameters and long training times. For instance, the 3-D U-Net has about 11 million parameters and took 3 days to train [Ref Ronneberger 3-D UNet]. In addition, deep convolutions encoded by NNs are intrinsically not rotationally invariant such that all rotations need to be explicitely learned and encoded by the NN. Especially in 3D this means that a lot of angles to be learned. As most biological data is rotationally invariant we feel that this is a disadvantage of NNs as compared with our approach where we only use rotationally invariant features.

- Here we chose to use fixed features, namely the eigenvalues of the hessian matrix and the structure tensor. These features have the advantage of being rotationally invariant and being good descriptors of most biologically relevant structures such as membranes, tubes, and vesicles.

- In neural network implementations of deep convolution the convolution kernels are learned by the NN during the training). However this results in many parameters to be learned and thus long training times (typically hours)).

- Computing features at higher resolution levels is not done by increasing the kernel width but by down-sampling the input image. This has the advantage of an increased speed during feature computation, as well as reduced memory requirements for storing the feature images. For N-D data with a binning factor of B, the reduction in computation time and memory is a factor of B^N for each resolution layer. For classification, the down-sampled feature images are up-sampled again (just as in the 3-D U-Net), this takes time such that some of the gain in speed is lost; we would like to explore in the future whether this up-sampling could be computed on a GPU in order to save time. As the up-sampling is only needed locally at the location of the current instance voxel, the additional memory requirements at this step are relatively small.

## Acknowledgements

We thank ...

## References

- ilastik

- fiji-tws

- imagescience.org

## Figure legends

### Figure\_Scheme

Schematic depiction of our deep convolutional feature random forest image segmentation algorithm. To go one resolution layer deeper, images are down-sampled by NxN average binning. At each resolution layer hessian matrix and structure tensor eigenvalue images are computed (HS) of all images coming from the previous resolution. For training and classification, binned images are upsampled using bilinear interpolation before they are fed into the random forest classifier. Widths of boxes indicate number of images, numbers inside boxes are the actual number of images for the example scenario of one 3-D input image. Heights of boxes indicate number of pixels per image.

### Figure\_LineOfDots

Examples of how deep convolution using hessian matrix and structure tensor combined with decision trees segments images.

A) Line of dots.

Supplemental Material

Down-sampling and feature-depth

For 3-D data, there are 3 ST and 3 HM eigenvalues for each input image, yielding in principle 7^L (1 input + 3 ST + 3 HM = 7 images) images at down-sampling level L. Since this is a very quickly growing number we decided to decouple the feature-depth D from the down-sampling level L. For instance, it is possible to choose 3 down-sampling levels at a maximal feature-depth of only 2

This, e.g., means that an image that already is the eigenvalue of an eigenvalue cannot be subjected to yet another eigenvalue computation at a lower resolution level. In our intuition this enables a compromise of covering many spatial length scales while keeping the number of feature images at bay.

### Feature importance and subsetting

The classical way to compute feature importances in RFs is to run an out-of-bag sample of the training data through the RF, and compare the classification results when one feature is exchanged by a random other feature [Ref Breiman]. As we typically have aroun2000 features and 400 trees, executing above recipe for all features takes a considerable amount of time and would thus perturb the interactivity of our tool. We thus opted for a different option: During the training we simply count how often each feature was used in the whole forest. The idea being that features which have been selected only at few nodes (in few trees) are probably not very important for the overall classification outcome. Such rarely used features can be deactived. In our current implementation this mainly speeds up the feature upsampling as this now needs to be done for less features. Currently we still compute all features (also deactivated) because features in later resolution levels are derived from features in earlier resolution levels such that it becomes somewhat involved to figure out which features can be left out during the feature computation stage. Moreover, after deactivating rarely used features we run the RF training once more, only taking into account the active features. Here our intuition is that the RF can learn more informative relations between actually useful features (TODO: test this somehow).

### Dealing with anisotropic data

The HM and ST feature computation algorithms are intrinsically dealing with anisotropic data [Ref\_ImageScience]. In addition, we deal with a potential anisotropy during the downsampling steps of our algorithm. For example, if the resolution of the input data is 200 nm in x/y and 600 nm in z, the first would downsampling - assuming a downsampling factor of 3 - would be 3x3x1 (instead of 3x3x3), yielding isotropic data with a (600 nm)^3 voxel size in the next resolution layer. The following binnings would be isotropic in this example.

### Random forest settings

A random forest has the following settings:

- N.. number of trees

- F.. number of random features per node

- ...

We chose F to be on tenth of the number of input features. Our intuition was that 1/10 is high enough to fetch the best features at each node with a decent probability, leading to a good classification strength of each tree, and low enough to have reasonably uncorrelated trees (the probability to have two sucessive nodes in two different trees using the same feature combination only is (1/10)^2 = 1/100). Both high strength and low correlation or important for a random forest classifier to work well [Ref Breiman].