Auto-context Feature Extraction for Neuron Segmentation

Ahmad Neishabouri
neishabouri@stud.uni-heidelberg.de

Marlen Runz
runz@stud.uni-heidelberg.de

Abstract

The process for training a classifier in image segmentation for computer vision, relies highly on how appropriate the image feature extraction methods used to train these classifier are. Although one could implement as many features available as possible. But there is no doubt some of them have a much higher contribution in producing the final probability map. In this work, we implemented a so called auto-context feature extraction algorithm on a biomedical image analysis problem by using Beiers et al. *[1] method of fusion moves for solving the NP-hard problem of optimizing the MAP.*

1. Introduction

“Similar as processing language, a single word may have multiple meanings unless the context is provided, and the patterns within the sentences are the only informative segments we care about. For images, the principle is the same. Find out the patterns and associate proper meanings to them.” [2] . Upon this fact, using context information, which means using the neighbourhood pixel information, is highly common in Computer Vision tasks([3], [4], [5]). Context is sort of looking at the whole picture, and finding spectral and spatial relations in the image [6]. For instance, one could always assume by looking for a tree, there should be a high frequency of green colour around it [7].

The objective of finding related context information is highly broad and there is a wide range of papers discussing different methods for context feature extraction. In this work we implemented the algorithm based on the paper from Z. Tu: Auto-context and Its Application to High-level Vision Tasks [8], named as a auto-context model.

1.1. Auto-context algorithm

The key spirit of this model is employing the probability maps produced from a learned classifier, for extracting new features, **context features**, and applying these new features to train another classifier, recursively. The procedure is as follows:

A classifier first will be trained on the set of training images and labels, with normal image features from local image patches. After the classifier been trained, it will be utilized on the same training set to produce predictions for them. This is where auto context comes into play, since these predictions are used to produce new features, the so called context-features. Along with the previously prepared local features, the new context features will now constitute another training set for the next classifier. This process will be repeated several times until the desired output is achieved. In the testing section, this procedure will be used in the same way that images have been trained, that is, the test image will go through all the classifiers one by one, and the context features will be produced accordingly, and ultimately the last classifier will result in the final image labeling.

1.2. Dataset

We applied this algorithm on the ISBI 2012 challenge for segmenting the neurons in the Drosophila larva ventral nerve cord images. The dataset contains 30 training images and 30 testing images. This work is a followup for Beier’s *et al.* [1] work on this challenge and we used their implementation as a reference for applying the auto-context algorithm. In order to avoid long time for grading and benchmarking, we used 80 percent of the dataset as training and the remaining as testing images.

2. Implementation

As we pointed out earlier, this paper applies an enhanced auto context algorithm inspired by Z.Tu’s [8] algorithm, on the ISBI neuron segmentation problem, and uses Beier’s *et al.* [1] implementation to do so. This work focuses mainly on two important areas. First, we apply a new dataset splitting scheme in which each set will have its own classifier and this classifiers will be trained sequentially. The second main enhancement would be the new problem-based context features extraction schemes. Both these enhancements are explained in this chapter.

To avoid repetition, we won’t discuss the overall implementation code, since it’s a follow up of Beier’s *et al.* [1] publication, and also, you can find the complete code they used under their Nifty library code documentation [9].

One important point that should be pointed out here is that in the course of doing this project, we realized that Beier's *et al*. [1], used a probability map, produced from their CNN network applied beforehand on the images in the set. These probability maps can enhance the result extensively, in a way that we couldn't see clear differences by our schemes. So we decided to not use them so that we could better see our implementation performance comparing to the original implementation.

2.1. Training the Classifiers

In contrast to the algorithm implemented in [8], we avoided using same data set for training the classifiers. Using the same training set over and over could result in over fitting, hence we planned to split the training set into desired numbers of sets. In this scheme, classifiers will be trained sequentially, meaning, the classifiers will be trained upon each other, and the last classifier will be the classifier of choice. Our classifier of choice was a Random Forest (RF) classifier provided by the Scikit-learn library [10]. To explain the procedure further, first a set of images will be used to train the first classifier. Then the algorithm will use this classifier, to extract context features, on the next image set, resulting a new training set with new features. This new training set will be used to train the next classifier. This classifier will then be used for the next images set and so on. Bare in mind that in all these processes, the ordinary image local features are extracted and attached to the new context features, simultaneously.

This scheme has been shown in Algorithm 1.

Algorithm 1 training

```
Data: trainDepth: number of training steps, imgSets: list
      containing arrays of image indices, feats: features for
      training/testing a classifier, predicts: edge probabili-
      ties computed by a classifier
Result: rf: list of trained RF classifiers
for $i \leftarrow 0$ to trainDepth do
    for img in imgSets do
        feats ← compute_image_feats(img)
        if $i \neq 0$ then
            predicts ← compute_edge_probs(rf[i-1], feats)
            feats ← compute_context_feats(predicts)
        end
        rf[i] ← train_classifier(feats)
    end
end
```

The code has been written in a way that any number of images set with any desired number of classifiers are possible to be used. In experiments section, we'll see the result of different combination of those two numbers and how changing them will result in different results. In case of small data set and high number of classifiers, a random image selection has been implemented to avoid small image sets.

2.2. Context feature extraction

As explained earlier, context features are those which are extracted from probability maps, resulted from the trained classifiers. We have extracted these features with three different schemes. Algorithm 2 shows the main steps of computing the context features.

Algorithm 2 compute_context_feats(edgeProbs)

```
Data: edgeProbs: edge probabilities computed by a classi-
      fier, regs: array of numbers used as regularizers for
      agglomerative clustering, sigmas: array of numbers
      used as sigmas for image filtering
Result: feats: features for training/testing a classifier
feats ← []
feats.append ← edgeProbs
feats.append ← agglomerative_clustering(edgeProbs, regs)
image ← img_from_probs(edgeProbs)
feats.append ← filtering(image, sigmas)
feats.append ← random_walker(image)
return feats
```

As can be seen, different algorithms where used for obtaining context features. First feature extractor we used was the agglomerative clustering and dendrogram heights. Second an image from edge probabilities was extracted which was modified by several image filtering techniques to obtain features. In the final step we implemented a random walker algorithm which was also applied on the image in order to extract features. All steps are explained in more detail in the following. It's not trivial to mention that our probabilities output itself was also a feature that we appended to our new feature set.

2.2.1 Agglomerative clustering and Dendrogram Heights

Hierarchical methods are methods in which they recursively try to cluster two items at a time. They follow two major schemes, first agglomerative and the second is the partitioning or the divisive scheme. In the partitioning method, starts from considering all the data points as one cluster and it start to partition them step by step into homogeneous parts until every cluster have identical points in each cluster. In agglomerative clustering however, this process starts from the point in which each item has it's own cluster and the algorithm tries to accumulate the similar clusters together one step a time. The agglomerate method is on of the important well established methods in unsupervised machine learning problem [11].

Depending on the similarity scheme used, hierarchical clustering algorithms fuse different objects to new clusters in which later on, the dissimilarity between the clusters will be displayed as a tree diagram called dendrograms (see Figure 1). These dendrograms help us visualize different clusters granularities. In addition to that, the important benefit of using Hierarchical clustering is the freedom of choosing any kind of distance metric for similarities [12]. This feature allowed us to use the predictions that we get from out classifiers as a similarity parameter for our clustering scheme.

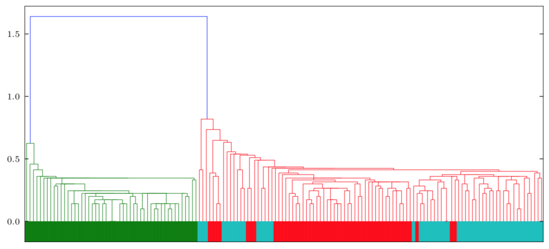


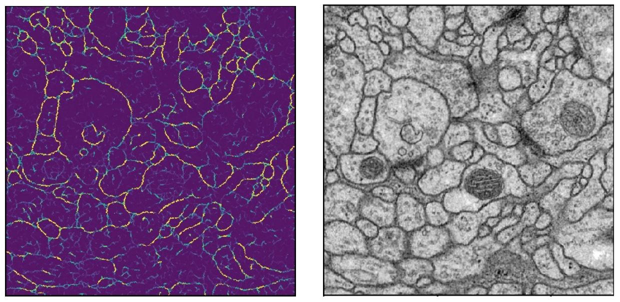
Figure 1: Example Single Linkage Dendrograms

We applied the mentioned scheme with different regularizers, ranging from 0.01 to 0.8 and appended the resulted features to our original feature set. The function we used for implementing the agglomerative hierarchical clustering is employed in the Nifty library and we used that for our feature extraction,

2.2.2 Visualization of the Probability map

In this method, we wrote a function in which we visualized the probability map produced from the classifier into an image with the same size of the original image. This function builds each pixel in the new image based on their neighbours, in a sense that, it goes over all the pixels in the original size image, check the neighbouring to see if they are from the same class label(super pixel), if they are not, it find the edge assigned to these two labels and read the predictions out from the early classifiers, and assigns this value to the both pixels. With this scheme, by going through all the pixels, will have an image like Figure 2 as a probability map for all the edges. By comparing it to the original image, it's clear that in the borders of the neurons, which they should be a high amount of edge probability, the pixels are bright, and the new visualized image can sort of resemble an edge extraction for the original image.

In our training, we have considered this new image as an input for our feature extraction system and by applying the same feature extraction to it, we'll have new features to augment to our original features. Clearly, this new features are the result of our probability map that we get from our



(a) Edge probabilitiy visualization (b) Raw data

Figure 2: Visualization of the edge probability map (a) compared to original raw data (b)

earlier classifier so we consider these features as context features for our training.

2.2.3 Random Walker

The random walker algorithm is a widely used algorithm [13] for image segmentation in which a user, interactively chooses a small group of pixels in an image as *seeds*, and then the algorithm well release random walkers from unlabeled pixels and the probability of each random walker to reach the seed will be computed. In this work however, we tried to apply this scheme as context features for our learning process.

Starting with our graph based image, in that each super pixel corresponds to a node and the neighbouring super pixels connected by edges. The edges are weighted by projecting the predictions acquired from our early classifiers. Since in our calculation, the feature extraction is edge wise(means the features are extracted upon the edges rather than the nodes/pixels), for each edge the starting seed will be one of the ends, and the end seed, the target, will be the other end. A random walker will repeatedly will be released from one end and we'll calculated the number of steps it takes to reach target. This kernel will be repeated for many times and in each time, the number of steps taken by the random walker will be calculated. At the end, this numbers will be averaged over the times that this kernel been repeated and a number will be assigned to that edge. This scheme will go over all the edges in the data set and will record a new feature for the corresponding edge. Eventually this will yield to a feature vector, attached to the other context feature. Figure 3 shows a histogram of number of walks based on probability binning. As it's showed in the figure, low probability edges take more walks to get to the target comparing to the higher probabilities.

As we explained earlier, since we stopped using the probability map that Beier's *et al.* [1] uses, our region adjacency is highly dense with over 5000 edge for each image. This would result a really heavy computations when we are

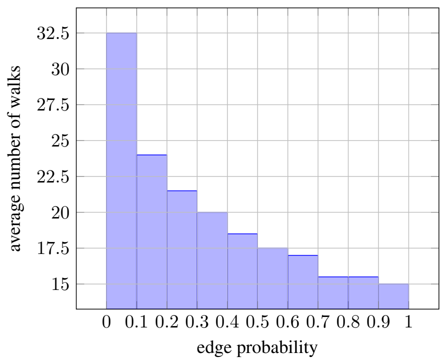


Figure 3: Histogram of number of walks by the edge probabilities

applying the random walker scheme. Moreover, the number of random walks can get very high in case the walker walks far away from the original seed, and clearly, when this walks get very large, it wouldn't represent any information to the edge probability any more. Moreover large number of walks can make our average very big and uncorrelated to the edge weight. Hence it's better to limit the number of walks to a finite number, like in this work, where it was limited to 250 walks.

2.3. Testing the classifier

As motioned earlier, for bypassing the bench marking process with the ISBI challenge coordinators, we kept one fifth of the database for bench marking purposes. This section has not been used in any of the training sets. Important to notice, that the testing stage should follow the same procedure as the training, in regard of producing the context features, since the new random forests are trained with new features and these new features should be build for the testing stage, as well. The last classifier will then give the final probability maps which from there, the same procedure of solving optimization problem [1] will be applied. This is done for each image as seen in Algorithm 3

Algorithm 3 testing

```
Data: trainDepth: number of training steps, testSets: list
      containing arrays of image indices, feats: features for
      testing a classifier, rf: list of trained RF classifiers
Result: predicts: edge probabilities computed by a classi-
        fier
for img in testSet do
    feats ← compute_image_feats(img)
    for i ← 0 to trainDepth do
        predicts[img] ← compute_edge_probs(rf[i],feats)
        feats ← compute_context_feats(img)
    end
end
```

3. Experiments and Results

The segmentation results are shown in Figure 4.

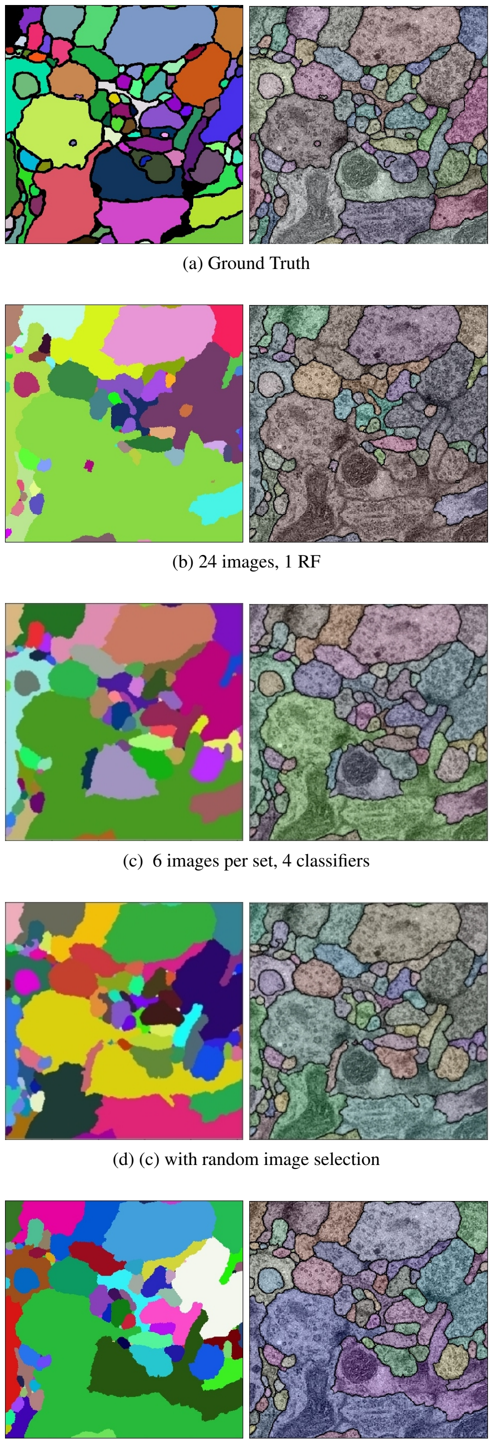
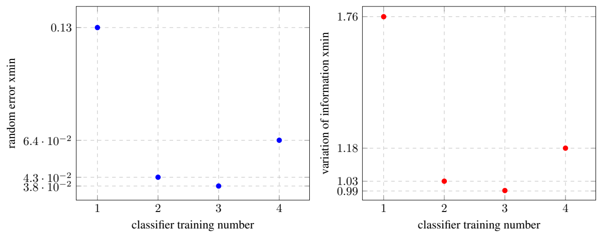

(a) Ground Truth
(b) 24 images, 1 RF
(c) 6 images per set, 4 classifiers
(d) (c) with random image selection
(e) (c) with Random Walker feature

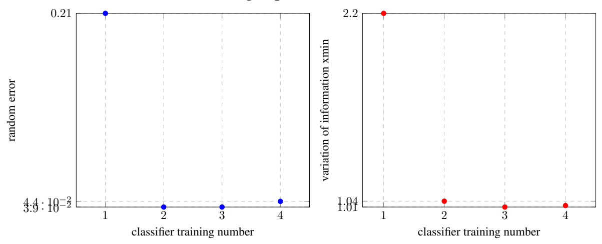
Figure 4: Neuron segmentation ground truth (a) and for different experiment setups (b)-(e)

As mentioned in the introduction section, by eliminating the step in which Beier's algorithm uses the convolutional neural network's probabilities, the results would deteriorate drastically. This can be seen by comparing Figure 4b with Figure 4a, the ground truth. Figure 4b is the result of Beier's algorithm, with ordinary local image patches features and without the CNN probabilities. It only has 1 random forest and our aim in this work was to enhance the results based on this output. In all our experiments, 20 percent of the data set left untouched for testing set. Moreover, since the random forest were heavily computational expensive, we included it separately in our output.

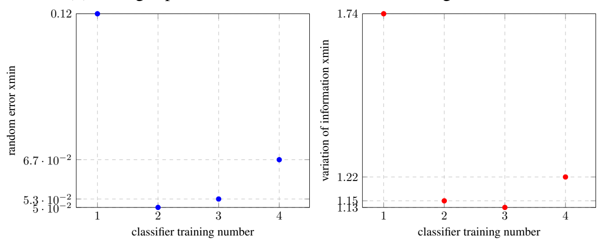
Figure 4c displays our output with using 4 image sets of 6, and incorporating 4 different random forest classifiers, in which each has been trained separately by the images sets. The context features has been fed sequentially from the previous classifier predictions and been appended to the training sets. This output is the prediction results of the 4th or respectively the last trained classifier. We can see some improvements both visually by comparing the result images, and by doing error calculations(Figure 5), as explained later. Figure 4c shows the enhanced classifier can detects the neurons better than the original one.



(a) 6 images per set, 4 classifiers



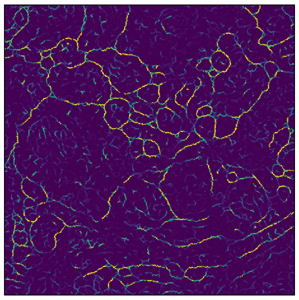
(b) 6 images per set, 4 classifiers with random image selection



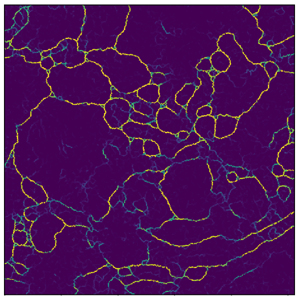
(c) 6 images per set, 4 classifiers with Random Walker feature

Figure 5: Random Error and Variation of Information for different experiment setups (a)-(d)

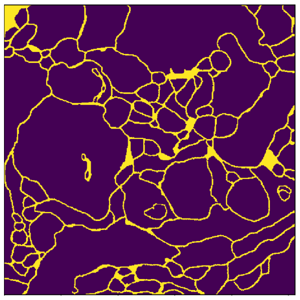
For evaluating the results we computed the random error by utilizing Scikit-learn [10] metrics package and the variation of information scheme. Figure 5 shows the resulted errors by each classifier. These errors has been calculated by comparing them to the ground truth. These diagrams clearly show that the main enhancement has been done by the second classifier and remaining classifiers won't contribute in reducing the error. We had the same observation, by running the code for 10 different random forests. In all the remaining random forests, the error oscillates up and down with no noticeable enhancement. Figure 4d was our best output, in which it employs the random image extraction option in our code. However this result is not consistence and it can varies, depending on the training set selection and the tested image used to employ.



(a) Without context features



(b) With context features



(c) Ground truth

Figure 6: Prediction map visualization for the two cases of with (b) and without (a) use of context features in training

In Figure 6, we have some interesting results, in which we used our visualization function that we designed for feature extraction for visualizing and comparing the resulted edge predictions. Figure 6a is the visualization of predictions for the case with no context feature and the use of only one classifier(the default no context case). As the image shows, there is high frequency of edges with medium intensity inside the neurons which definitely would result in miss classification error in our classifier, however after the use of context features in our training(Figure 6b), the predictions are a lot more close to the ground truth and the extra medium range edges have now close to zero intensities.

Figure 4e shows our output with using of our random walker feature extraction option. There is a wide variety of ways to run the random walker kernel in our code. One can run the random walker exhaustively, in which the kernel walks until it finds the target, or one can put number of walk limit to the search. We realized the best way to do it, is when the histogram of the intensities perfectly resembles the edge probabilities(Figure 3. The limitation used for this simulation was 250 walks between two nodes, and the average of walks was taken after 50 repetition of the search.

References

[1] *An Efficient Fusion Move Algorithm for the Minimum Cost Lifted Multicut Problem*, volume LNCS 9906. Springer, 2016.

[2] Godfried T. Toussaint. The use of context in pattern recognition. *Pattern Recognition*, 10(3):189 – 204, 1978. The Proceedings of the IEEE Computer Society Conference.

[3] A context algorithm for pattern recognition and image interpretation. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(30):189 – 204, 1971.

[4] Zhuowen Tu and Xiang Bai. Exploiting human actions and object context for recognition tasks. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999.

[5] Antonio Torralba, Kevin P. Murphy, William T. Freeman, and Mark A. Rubin. Context-based vision system for place and object recognition computer vision. *IEEE International Conference on In Proceedings Ninth IEEE International Conference on Computer Vision*, 2003.

[6] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, April 2002.

[7] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Putting objects in perspective. *Int. J. Comput. Vision*, 80(1):3–15, October 2008.

[8] Zhuowen Tu and Xiang Bai. Auto-context and its application to high-level vision tasks and 3d brain image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(10):1744–1757, October 2010.

[9] Nifty 0.19.0 documentation. `http://derthorsten.github.io/nifty/docs/python/html/index.html`, 2017. [Online; accessed 15-August-2017].

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

[11] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[12] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, Sep 1967.

[13] Leo Grady. Random walks for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(11):1768–1783, November 2006.