Guide to segmentation of tissue images using MATLAB script with Fiji and Weka

Zhang Chuheng
zhangchuheng123@live.com

September 3, 2015

1 Overview

This guide demonstrates a machine learning approach to do segmentation on biological images. Here, we use mainly ssTEM images of neural tissue, but this approach is easy to transform into segmenting other biological images. This is a guide showing how to select training points from labeled image, gather feature stack of the training points, train the classifier and apply the classifier to images utilizing open-source package Fiji with Weka. Post-processing is always needed after pixel classification. Hence, a little about post-processing is also mentioned in this guide.

As is shown in figure 1, the whole process involves three parts. The first part is the training of pixel classifier which includes selecting training points, obtaining features of training points and the the training of classifier. .arff files can be produced after the feature stack of selected training points is created which can be used as an input of training classifier. .model files can be produced to store the information of classifier and can be applied directed to test images. The second part is applying the classifier to test images. The third part involves some post-processing which takes more knowledge about different parts of neural tissue, in this case, into account. Here, I considered the neighbors of each pixel, the connectivity of the intercellular substance/ membranes. Additionally, smoothing of edges are also applied to the segmented images.

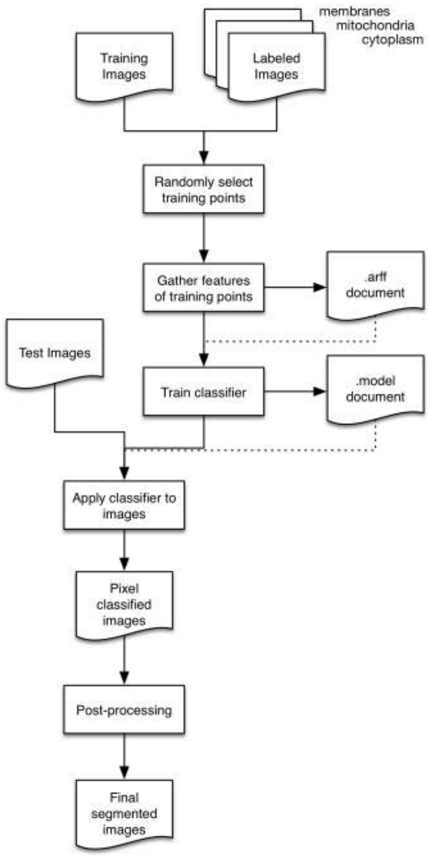

Figure 1: The process of biological image segmentation

2 Details of the process

2.1 Configuration

The relevant softwares/packages that I use are MATLAB 8.3.0.532 (R2014a), ImageJ 2.0.0-rc-34/1.50a and Java 1.8.0-25(64-bit).

Fiji provided an interface for MATLAB. In order to use it, steps below should be followed.

1. Add the path of Fiji-MATLAB interface (Miji.m) to MATLAB working path.

   execute `addpath('<Fiji.app>/scripts');` in MATLAB command window (Mac OS), or `addpath(fullfile('<Fiji.app>','scripts'));` in MATLAB command window (All platform), whereas <Fiji.app> represent for the root of Fiji on your computer. Both relative or absolute root are valid here.

2. In order to utilize the java packages that Fiji provides, `Miji;` should be executed in MATLAB command window. A Fiji UI will prompt up and DO NOT close that window and leave it there.

3. In order to utilize the java packages that Trainable Weka Segmentation provides, `MIJ.run('Trainable Weka Segmentation');` should be executed in MATLAB command window. The UI of Trainable Weka Segmentation will prompt up and DO NOT close that window and leave it there.

One thing should be noticed is that you can put the three commands above in your script, but make sure those three commands should be ONLY executed ONCE on each launch of your MATLAB. Hence, to execute them in MATLAB command window is recommended.

Now, your computer is well-prepared for image segmentation with Fiji and Weka.

In some cases during the running of your program, errors like `java.lang.OutOfMemoryError: Java heap space` will occur. You may increase the java heap space allowed in MALAB by performing following operations on your computer. Set up an ASCII file, write `-Xmx2048m`, save the file as `java.opts` and put the file to `$MATLABROOT/bin/$ARCH` or the directory that is shown when you execute `pwd` in the command window of MATLAB. For more information on how to increase the java heap space in MATLAB, click here.

2.2 Preparation of image files

All the images and segmented labels of the images are downloaded from here which are Segmented anisotropic ssTEM dataset of neural tissue by Stephan Gerhard, et al.

Put the MATLAB scripts you have written and the images files in the MATLAB working directory as shown in figure 2.

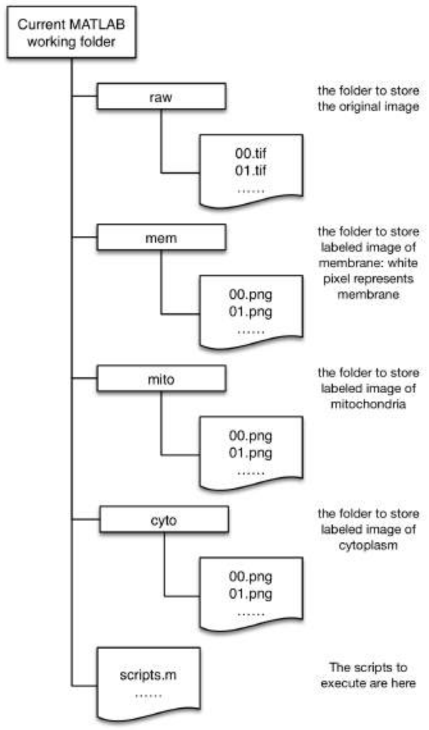
Current MATLAB working folder
raw
the folder to store the original image
00.tif
01.tif
......
mem
the folder to store labeled image of membrane: white pixel represents membrane
00.png
01.png
......
mito
the folder to store labeled image of mitochondria
00.png
01.png
......
cyto
the folder to store labeled image of cytoplasm
00.png
01.png
......
scripts.m
......
The scripts to execute are here

Figure 2: Directory structure

There are different ways in which images can be store. The labeled images can be stored as floating-point number between 0 and 1 where 0 represents background pixels and 1 represents foreground pixels. However, image reader of ImageJ (java method ij.IJ.openImage) can only recognize, from my point of view, images stored in another way in which unsigned integers are used to represent pixels. 0 represents background pixels and 1 represents foreground pixels (8-bit image). Hence, following script can be used to make sure that the images are in correct formats. The script will not change anything if the label images are already in a correct encoding format.

Listing 1: preparation of label images

```
%% INFO
% Author: Zhang Chuheng
% Email: zhangchuheng123@live.com
% Date: 12/08/2015
```

```
%% DESCRIPTION
% This file is to turn all the logic TIF/PNG files into a uint8 TIF/PNG
% files - the format Tainable Weka Segmentation can recognize.

for i = 0:19
    filename = sprintf('%.2d',i);

    % rewrite the label of membranes in correct format
    f_mem = imread(['./mem/',filename,'.png']);
    f_mem = uint8(logical(f_mem)) .* 255;
    imwrite(f_mem,['./mem/',filename,'.png'],'png');

    % rewrite the label of mitochondria in correct format
    f_mito = imread(['./mito/',filename,'.png']);
    f_mito = uint8(logical(f_mito)) .* 255;
    imwrite(f_mito,['./mito/',filename,'.png'],'png');

    % The dataset did not provide label image of cytoplasm,
    % so I generate the label of cytoplasm from the complementary
    % image of mitochondria and membrane.
    f_cyto = imcomplement(imadd(f_mito,f_mem));
    imwrite(f_cyto,['./cyto/',filename,'.png'],'png');
end
```

2.3 Training of classifier and pixel classification of images

Here, the classifier is trained in three steps. First, several training points of each class are randomly selected and the feature stack of these points are created and saved to .arff files. Second, several .arff files are selected as the training input and merged into one. Third, that one .arff file is used for training and a classifier will be trained. You can either save that classifier to a .model file or apply that classifier to other images.

2.3.1 Point selection and feature extracting

ImageJ class (`ij.IJ`) are used to read images and the images will be store as an instance of ImagePlus. An instance of WekaSegmentation will be created from this ImagePlus object by
`segmentator = trainableSegmentation.WekaSegmentation( trainInput );`
which equals to
`segmentator = new trainableSegmentation.WekaSegmentation( trainInput );`
in java. You can always translate Java into MATLAB like this. Pleas click here for more information on how to call Java libraries in MATLAB. `setEnableFeatures` can be used to set features that should be extracted from the image and `setMaxiumSigma`

can be used to set the maximum window size in feature extracting. addRandomData is used to randomly select points on the white pixels of label images and extract features of those points. This function, in general, will take a long time. Please click here to look up all the methods of WekaSegmentation you can use.

The code is listed below.

Listing 2: Select points and create feature stack of those points

```
%% INFO
% Author: Zhang Chuheng
% Email: zhangchuheng123@live.com
% Date: 11/08/2015

%% DESCRIPTION
% This script is used to generate a ARFF file for weka image segmentation
% training. The original training image and its label files are needed as
% the input. The core of weka segmentation will be called to randomly
% select points of the label files and generate the attributes of those
% points.

%% ATTENTION
% However the approach is quite UNSTABLE. You should check the validation
% manually or use F3.m in the suite to check.

%% PREREQUISITE
% You should at first run following codes in the command window of MATLAB
% ONE BY ONE.

% addpath('/Applications/Fiji.app/scripts')
% %This is the typical path of Fiji on Mac OS, you may change it to the
% correct path of Fiji on your platform

% Miji;
% % Do NOT close the main menu of FIJI

% MIJ.run('Trainable Weka Segmentation');
% % randomly select a picture to open and DO NOT close the window of
% TRAINABLE WEKA SEGMENTATION prompted up

%% Troubleshooting
% You may encounter problems like 'java.OutOfMemory'. You can allocate more
% heap space to java by writing '-Xmx2048m' to a file named 'java.opts' to
% .../Documents/MATLAB/ folder

%%
import ij.*;
import trainableSegmentation.*;
import trainableSegmentation.WekaSegmentation;
import trainableSegmentation.Weka_Segmentation;
```

```
numOfImages = 20;

for i=1:numOfImages
      num = sprintf('%.2d',i-1);
      trainInput = IJ.openImage(['./raw/',num,'.tif']);
      trainLabelMito = IJ.openImage(['./mito/',num,'.png']);
      trainLabelMem = IJ.openImage(['./mem/',num,'.png']);
      trainLabelCyto = IJ.openImage(['./cyto/',num,'.png']);

      segmentator = trainableSegmentation.WekaSegmentation( trainInput );

      segmentator.addClass();

      enableFeatures = [
                  true,  ...%/* Gaussian_blur */
                  true,  ...%/* Sobel_filter */
                  true,  ...%/* Hessian */
                  true,  ...%/* Difference_of_gaussians */
                  true,  ...%/* Membrane_projections */
                  true,  ...%/* Variance */
                  true,  ...%/* Mean */
                  true,  ...%/* Minimum */
                  true,  ...%/* Maximum */
                  true,  ...%/* Median */
                  true,  ...%/* Anisotropic_diffusion */
                  true,  ...%/* Bilateral */
                  true,  ...%/* Lipschitz */
                  true,  ...%/* Kuwahara */
                  true,  ...%/* Gabor */
                  true,  ...%/* Derivatives */
                  true,  ...%/* Laplacian */
                  true,  ...%/* Structure */
                  true,  ...%/* Entropy */
                  true   ...%/* Neighbors */
      ];

      segmentator.setEnabledFeatures( enableFeatures );

      segmentator.setMaximumSigma(64.0);

      numOfTrainingPoints = 400;

      segmentator.addRandomData(trainInput,trainLabelMem,'class 1',...
            numOfTrainingPoints);
      segmentator.addRandomData(trainInput,trainLabelMito,'class 2',...
            numOfTrainingPoints);
      segmentator.addRandomData(trainInput,trainLabelCyto,'class 3',...
            numOfTrainingPoints);
      segmentator.writeDataToARFF(segmentator.getLoadedTrainingData(),...
            ['Features-',num2str(numOfTrainingPoints),'-',num,'.arff'] );
```

```
end
```

2.3.2 Training data selection and merging

Several .arff files are selected as training input and put into the folder named ArffFolder. This folder is of the same level and place as folder raw, mem, etc. that we used just now to store images. An output file named data.arff will be produced by the script. There are also methods to merge several .arff files by using Weka Trainable Segmentation API.

This code is listed below.

Listing 3: Merge .arff files in ArffFolder into one

```
%% INFO
% Author: Zhang Chuheng
% Email: zhangchuheng123@live.com
% Date: 12/08/201

%% DESCRIPTION
% This script is used to merge several ARFF files into one. It will check
% whether the individual ARFF files are correct and whether the number of
% attributes in each ARFF file are equal.

%% followings are the parameter that you can modify
% the folder that holds the seperate ARFF files
folder = 'ArffFolder';
% the name of output file
outputName = 'data.arff';

% open the folder
list = dir(fullfile('.',folder));
% delete the outputfile first if exist
delete(outputName);
% count the training points of each class
count = [0,0,0];
fout = fopen(outputName,'w');
% whether or not the data part of the first file begins
isBegin = false;
for i = 1:length(list)
    name = list(i).name;
    % in Mac OS there will be some system items in the folder begining with
    % '.'. This is to ignore them.
    if strcmp(name(1),'.')
        continue;
    end
    fid = fopen(fullfile('.',folder,name),'r');
    % to count how many attributes this .arff file has
    linenum = 1;
    line = fgetl(fid);
    first = true;
    while ischar(line)
        % ignore the lines that begin with '@' in the arff file
        if (isempty(line)) || (line(1) == '@')
            if ~isBegin
                % to put the head of the first file into the output file
                fprintf(fout,'%s \n',line);
            end
            line = fgetl(fid);
            linenum = linenum + 1;
        else
            isBegin = true;
            % Check if the attributes of the arff are all the same
            % by displaying in which line does the data part start.
            % It should not be correct if data part starts at
            % different line in different files.
            if first
                disp(['Data of file ',name,' starts at ',num2str(linenum)]);
                first = false;
            end

            % count the number of training points of each class
            if strcmp(line(end-7:end-1),'class 1')
                count(1) = count(1) + 1;
            elseif strcmp(line(end-7:end-1),'class 2')
                count(2) = count(2) + 1;
            elseif strcmp(line(end-7:end-1),'class 3')
                count(3) = count(3) + 1;
            else
                % a frequent error: the name of the class is not 'class X' but
                % just a number
                disp('error');
                pause();
            end
            % print the line to the output file
            fprintf(fout,'%s \n',line);
            line = fgetl(fid);
        end
    end
    fclose(fid);
end
fclose(fout);


disp(['The number of instances in class ',num2str(num(1)),...
    ' is ',num2str(count(1))]);
disp(['The number of instances in class ',num2str(num(2)),...
    ' is ',num2str(count(2))]);
disp(['The number of instances in class ',num2str(num(3)),...
    ' is ',num2str(count(3))]);
```

2.3.3 Classifier training

Here, the features of training points are loaded from an .arff file and are used for training. Later, the trained classifier is stored in a .model file. You can later load the .model file and apply the classifier to images or you can do these two things together.

The code is listed below.

Listing 4: Train the classifier from .arff file and save trained classifier into a .model file

```
%% INFO
% Author: Zhang Chuheng
% Email: zhangchuheng123@live.com
% Date: 21/08/2015

%% DESCRIPTION
% This script is used to generate a ARFF file for weka image segmentation
% training. The original training image and its label files are needed as
% the input. The core of weka segmentation will be called to randomly
% select points of the label files and generate the attributes of those
% points.

%% ATTENTION
% However the approach is sometimes UNSTABLE. You'd better check the
% validation of the output file, or you can use F3.m which checks
% frequent mistakes.

%% PREREQUISITE
% You should at first run following codes in the command window of MATLAB
% ONE BY ONE.

% addpath('/Applications/Fiji.app/scripts')
% %This is the typical path of Fiji on Mac OS, you may change it to the
% correct path of Fiji on your platform

% Miji;
% % Do NOT close the main menu of FIJI

% MIJ.run('Trainable Weka Segmentation');
% % randomly select a picture to open and DO NOT close the window of
% TRAINABLE WEKA SEGMENTATION prompted up

%% Troubleshooting
% You may encounter problems like 'java.OutOfMemory'. You can allocate more
% heap space to java by writing '-Xmx2048m' to a file named 'java.opts' to
% .../Documents/MATLAB/ folder
```

```
%%
import ij.*;
import trainableSegmentation.WekaSegmentation;
import trainableSegmentation.Weka_Segmentation;

trainInput = IJ.openImage(fullfile('.','raw','00.tif'));
segmentator = trainableSegmentation.WekaSegmentation(trainInput);

%% make the class names corresponding to the class names in the ARFF file
% by ADDCLASS the name of the new class is 'class 3'
segmentator.addClass();

%% load data and train
segmentator.loadTrainingData('data.arff');
segmentator.trainClassifier();
segmentator.saveClassifier('classifier.model');
```

2.3.4 Pixel classification of test images

.model file is loaded and applied to each image from the image stack. A file saver is instantiated to save the result as image files. Before loading training data we have to use `addClass();` to ensure that the number and the name of the classes are corresponding to those in the training data(.arff file). However, you do not need to do that before you load a classifier(.model file).

Listing 5: Load the classifier from .model file and apply it to images

```
%% INFO
% Author: Zhang Chuheng
% Email: zhangchuheng123@live.com
% Date: 13/08/2015

%% DESCRIPTION
% This script is used to generate a ARFF file for weka image segmentation
% training. The original training image and its label files are needed as
% the input. The core of weka segmentation will be called to randomly
% select points of the label files and generate the attributes of those
% points.

%% PREREQUISITE
% You should at first run following codes in the command window of MATLAB
% ONE BY ONE.

% addpath('/Applications/Fiji.app/scripts')
% %This is the typical path of Fiji on Mac OS, you may change it to the
% correct path of Fiji on your platform

% Miji;
% % Do NOT close the main menu of FIJI

% MIJ.run('Trainable Weka Segmentation');
% % randomly select a picture to open and DO NOT close the window of
% TRAINABLE WEKA SEGMENTATION prompted up

%% Troubleshooting
% You may encounter problems like 'java.OutOfMemory'. You can allocate more
% heap space to java by writing '-Xmx2048m' to a file named 'java.opts' to
% .../Documents/MATLAB/ folder

%%
import ij.*;
import trainableSegmentation.WekaSegmentation;
import trainableSegmentation.Weka_Segmentation;

for i = 1:20
    %% open the existed ARFF data
    % the data can be any existed valid arff file, but pay attention the class
    % names in the ARFF file should be corresponding to the wekaSegmentation
    % instance.

    % you can modify the name of image that you want to modify
    num = sprintf('%.2d',i-1);
    trainInput = IJ.openImage(fullfile('.','raw',[num,'.tif']));
    segmentator = trainableSegmentation.WekaSegmentation( trainInput );

    %% load classifier and train
    % you can modify the file name of the classifier you want to use
    segmentator.loadClassifier('classifier.model');

    %% classify image
    % use 'false' to get segemented image instead of probability map
    segmentator.applyClassifier( false );
    result = segmentator.getClassifiedImage();
    % save the image to files
    saver = ij.io.FileSaver(result);
    saver.saveAsTiff(fullfile('.','classifiedImage',[num,'.tif']));
end
```

2.4 Post-processing

The output image from pixel classifier is ragged. It looks not so good and can hardly be applied for further utilization. Generally, the output image looks like what is shown in figure 3. It is ragged and some isolated spots exist. That is because the process is just pixel classification. Hence, the classifier will not consider much about the connectivity of the pixel as well as the final labels of the neighborhoods. However, we can apply our knowledge about the tissue to the images to reduce the error rate. That the mitochondria are round shape and that the membranes are always connected to each other can be used to refine the segmented images.

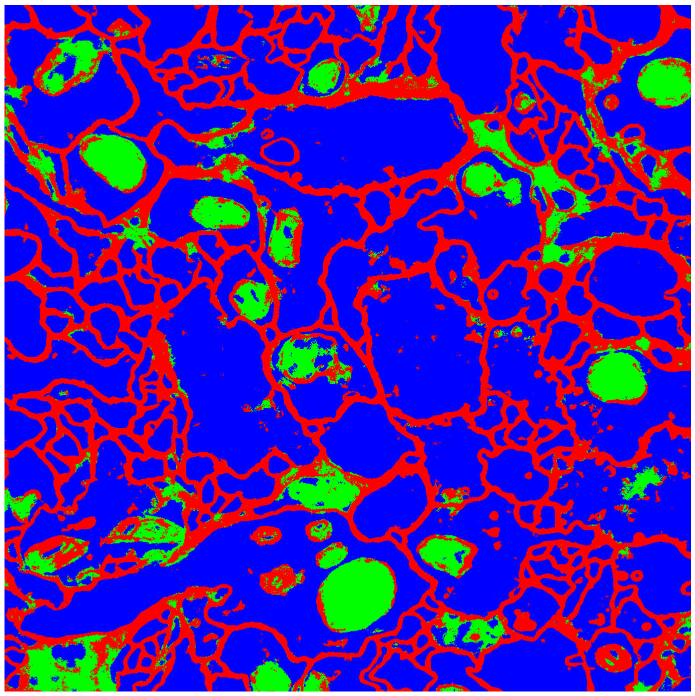
Figure 3: The image from pixel classification

2.4.1 Post-processing to reduce error rate

Here presented the procedure that I used to do post-processing. First, I removed the isolated small spots in the cytoplasm which makes the image clean. Then I corrected some obvious errors of misclassification of membranes as mitochondria or mitochondria as membranes utilizing connectivity and roundness. The image after the post processing looks like what is shown in figure 4.

However, the thresholds and even the methods of post-processing in this special case may not be effective in other cases. Hence, the script below is hard to apply directly to other cases. Although it varies from case to case the script below is a good example to illustrate how to do post-processing.

Listing 6: Post-processing

```
%% INFO
% Author: Zhang Chuheng
```

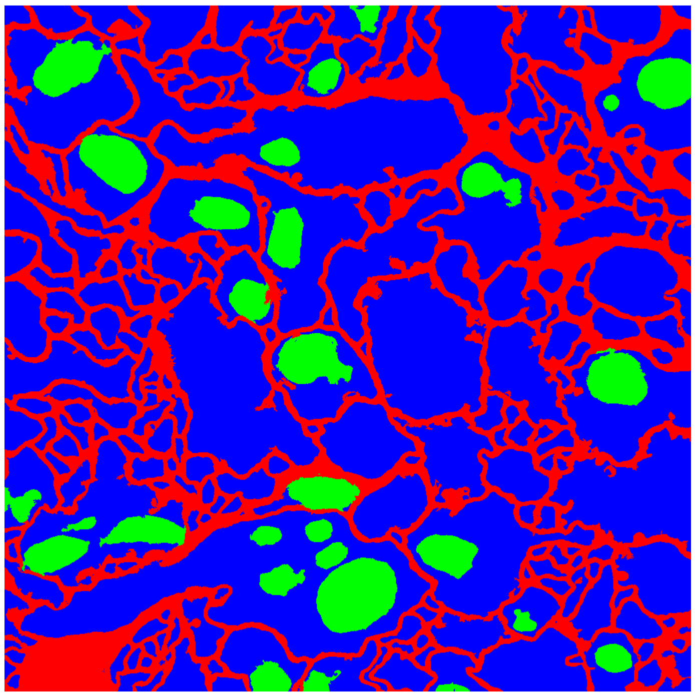
Figure 4: The image after post-processing

```
% Email: zhangchuheng123@live.com
% Date: 19/08/2015

%% DESCRIPTION
% Post-processing: the first part

%%
inputFolder = 'classifiedImage';
outputFolder = 'post1';
list = dir(fullfile('.',inputFolder));
map = [1,0,0;0,1,0;0,0,1;1,1,0];

for i = 1:length(list)
    name = list(i).name;

    % in Mac OS there will be some system items in the folder begining with '.'
    % This is to ignore them.
    if strcmp(name(1),'.')
        continue;
    end

    % mem-->0 mito-->1 cyto-->2
    % this is defined by the sequence of classes in previous process
    f0 = imread(fullfile('.',inputFolder,name));
    mem_mito = (f0 == 0) | (f0 == 1);

    %% This section is to remove the small isolated parts on cytoplasm
    cc = bwconncomp(mem_mito,4);
    s = regionprops(cc,'Area');
    area = cat(1,s.Area);
    % 400 below is the threshold of 'small'
    ind = find(area > 400);
    label = labelmatrix(cc);
    S_mem_mito = zeros(size(label));
    for k = 1:length(ind)
        S_mem_mito = S_mem_mito | (label == ind(k));
    end

    cyto = imcomplement(S_mem_mito);
    mem = (f0 == 0) & S_mem_mito;
    mito = (f0 == 1) & S_mem_mito;

    f0 = mem.*0 + mito.*1 + cyto.*2;

    %% This section is to refine the stucture or boundaries of mitochondria
    % and membranes using knowledge on the shape and size of the mitochondria.
    mem = f0 == 0;
    mito = f0 == 1;

    cc = bwconncomp(mito,4);
    s = regionprops(cc,'Area','Perimeter');
    area = cat(1,s.Area);
    peri = cat(1,s.Perimeter);
    ratio = area ./ (peri .* peri);
    ind1 = find(ratio < 0.01);
    ind2 = find(area < 400);
    ind = union(ind1,ind2);
    label = labelmatrix(cc);
    S = zeros(size(label));
    for k = 1:length(ind)
        S = S | (label == ind(k));
    end
    mem = imadd(mem,S);
    mito = imsubtract(mito,S);
    cyto = imcomplement(imadd(mem,mito));

    cc = bwconncomp(mem,4);
    s = regionprops(cc,'Area');
    area = cat(1,s.Area);
    ind = find(area == max(area));
    label = labelmatrix(cc);
    S = double(label == ind);
    mito = imadd(mito,imsubtract(mem,S));
    mito = imfill(mito,'holes');
    mem = S;
    cyto = imcomplement(imadd(mito,mem));

    % index starting with 0 is invalid for displaying with colormap
    % here, mem-->1 mito-->2 cyto-->3
    % instead of, mem-->0 mito-->1 cyto-->2, in the above
    f0 = mem.*1 + mito.*2 + cyto.*3;
    % save the output image files after post-processing
    % saving tif image by 'imwrite' will be moved to class TIFF in
    % future versions of MATLAB
    imwrite(f0,map,fullfile('.',outputFolder,[name(1:end-4),'.png']));
end
```

2.4.2 Smoothing

Although the different parts of the biological images are correctly segmented after post-processing listed above, the edges of each parts are ragged. Here presents the way that can be used to smooth the edges. The method of polynomial fitting is adopted to smooth the edge.

Firstly, bwboundaries is called to find the point series of boundaries. Next, x and y coordinates of the point series are regarded as one-dimensional data series separately for polynomial fitting. One thing worth mention is that `smoothX = cat(1, smoothX, transpose(linspace(smoothX(1), smoothX(end), 5)));`

is written to put more points between the start and end points to make sure that boundaries after smoothing are all closed.

Listing 7: Smoothing

```
%% INFO
% Author: Zhang Chuheng
% Email: zhangchuheng123@live.com
% Date: 20/08/2015

%% DESCRIPTION
% Post-processing: smoothing

%%
inputFolder = 'post1';
outputFolder = 'post2';
list = dir(fullfile('.',inputFolder));
map = [1,0,0;0,1,0;0,0,1;1,1,0];

for i = 1:length(list)
    name = list(i).name;

    if (strcmp(name(1),'.'))
        continue();
    end

    % mem-->0 mito-->1 cyto-->2
    f0 = imread([inputFolder,name]);

    %% smooth the boundaries of mitochondria
    mito = f0 == 1;
    boundaries = bwboundaries(mito);
    numberOfBoundaries = size(boundaries, 1);
    windowWidth = 11;
    polynomialOrder = 1;
    S_mito = zeros(size(f0));
    for k = 1 : numberOfBoundaries
        thisBoundary = boundaries{k};
        x = thisBoundary(:, 2);
        y = thisBoundary(:, 1);
        if (size(x,1) >= windowWidth)
            smoothX = sgolayfilt(x, polynomialOrder, windowWidth);
            smoothY = sgolayfilt(y, polynomialOrder, windowWidth);
            smoothX = cat(1,smoothX,...
                transpose(linspace(smoothX(1),smoothX(end),5)));
            smoothY = cat(1,smoothY,...
                transpose(linspace(smoothY(1),smoothY(end),5)));
            XX = round(smoothY); XX = XX + (XX == 0) - (XX > size(f0,1));
            YY = round(smoothX); YY = YY + (YY == 0) - (YY > size(f0,1));
            ind = sub2ind(size(f0),XX,YY);
            S_mito(ind) = 1;
        end
    end
    S_mito = imfill(S_mito,'holes');

    %% smooth the boundaries of membranes
    mem = f0 == 0;
    boundaries = bwboundaries(mem);
    numberOfBoundaries = size(boundaries, 1);
    windowWidth = 11;
    polynomialOrder = 1;
    S_mem = zeros(size(f0));
    for k = 1 : numberOfBoundaries
        thisBoundary = boundaries{k};
        x = thisBoundary(:, 2);
        y = thisBoundary(:, 1);
        if (size(x,1) >= windowWidth)
            smoothX = sgolayfilt(x, polynomialOrder, windowWidth);
            smoothY = sgolayfilt(y, polynomialOrder, windowWidth);
            smoothX = cat(1,smoothX,...
                transpose(linspace(smoothX(1),smoothX(end),5)));
            smoothY = cat(1,smoothY,...
                transpose(linspace(smoothY(1),smoothY(end),5)));
            XX = round(smoothY); XX = XX + (XX == 0) - (XX > size(f0,1));
            YY = round(smoothX); YY = YY + (YY == 0) - (YY > size(f0,1));
            ind = sub2ind(size(f0),XX,YY);
            S_mem(ind) = 1;
        end
    end

    %%
    % the following codes are used to select the membrane that is
    % outlined by S_mem
    S_mem = imcomplement(S_mem);
    cc = bwconncomp(S_mem,4);
    s = regionprops(cc,'Area');
    area = cat(1,s.Area);
    ind = find(area == max(area));
    label = labelmatrix(cc);
    S_mem = double(label == ind);

    %% Reassemble the image
    S_cyto = imcomplement(imadd(S_mito,S_mem));
    f0 = S_mem.*1 + S_mito.*2 + S_cyto.*3;

    %% save the image to file
    imwrite(f0,map,fullfile('.',outputFolder,[name(1:end-4),'.png']));
end
```

3 Possible future improvement on segmentation of images

There are some deficiencies of the final image of the segmentation. The out of bag error of pixel classification reaches up to 18%. Some of the misclassified pixels are causing big trouble for post-processing. Here proposed some possible ways to improve the quality of the segmented image.

3.1 Increase the window size

By increasing the window size in the process of feature extraction, more surrounding pixels will be considered and the classification will be more accurate. You use the method `setMaximumSigma(<float>)` in the class wekasegmentation to set the window size. Bigger the window is, more accurate the results will be. Nevertheless, bigger window size leads to more features extracted and more features leads to more memory and time in calculation. In the shown case, the window size, also known as sigma, is 64.0 which reaches the limit of my laptop. However, larger window size is preferred if computing capability allowed.

Increasing the window size has its basis. By increasing the window size from 16.0 to 64.0, bigger mitochondria of the size around 50 pixels are recognized. Some big mitochondria whose sizes are more than 70 pixels are missing and they are expected to be recognized by increasing the window size.

3.2 Select more representative points

In this example, we use the method `addRandomData()` provided by wekasegmentation to select point randomly. This method, however, is not a efficient way of selecting points. Selecting points that are near the boundaries of different parts is proposed.

The reason is as follows. The points that are on the center of each part is obviously the most typical points. However, including large number of such points to training data is a kind of consuming computing power since those points have little help in deciding the decision boundary in machine learning. (Shown in figure 5 a.) If points near the boundary is more intensely selected, better result will be obtained followed by more accurate decision boundary. (Shown in figure 5 b.) That will also reduce the number of training point needed to find out similar decision boundary. Admittedly, selecting points near the boundary will increase the out of bag error rate, but the result will improve. Additionally, in this way, the points on the boundaries will be better classified which helps a lot for later post-processing.

I have tried manually selecting the points near the edges, and apparent improvement was observed, which verified the effectiveness of the proposed method. Furthermore, selecting points near the edges is feasible. The points can be picked up on the parts which are the original parts subtracted be the eroded parts.

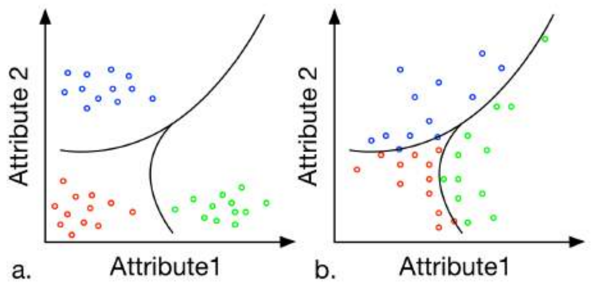

Figure 5: Two ways of selecting training points.
a. randomly select point. b. select points near the boundaries

3.3 Select features more carefully

Definitely, some features have more power to distinguish one class from the other than other features. It is suggested that this kind of features should be selected or produced, whereas the features help little in telling the parts apart should be eliminated. The features should be better understood and selected to promote the efficiency of the the classifier.

3.4 Increase the scale of training data

In the region of machine learning, the bigger the training data, the better the results. Do not do that, however, unless you have no more space improving accuracy and efficiency of the classifier. Nevertheless, in this special case, there are plenty space increasing the scale of training data. The number of total points in the 20 test figures is $1024 \times 1024 \times 20(images) = 2 \times 10^7$, whereas 4 figures (00.tif, 05.tif, 10.tif, 15.tif) are selected as training image but only 400 points are selected as training data. Hence, the number of the points in training data is $400 \times 4 = 1600$.

3.5 Improve post-processing

The post-processing is the most trivial part but also a indispensable component. Consideration on connectivity, the expected shape and the size, etc. is always helpful.

4 Afterword

This approach involves little about specific characters of the given tissue, so it is easy to be transformed into segmenting other types of tissue that OBEL may encounter in the future. I hope it may be helpful in your future research.

Words cannot express how much I appreciate to be with you, especially fruitful discussions with Philip and Andrea, insightful suggestions from David, valuable advice from Robert, generous and all-round help from Peijun, powerful technical support from Rodney and wonderful time with the two other Chinese interns Jingjie and Guxin. Thank you all so much and I really enjoy to be with you.