**Using SVM to detect EPTA-stained synapses in electron microscopy images**

Features\_label\_SRE\_125\_61.mat contains –

1. Features and Labels of 1665 synapses and 10441 non-synapse objects that were manually labeled.
2. Model\_SRE\_61 is the classifier built from these features and used for synapse detection. To build a classifier from scratch, check out the section ‘Building your own Model’ below.

Building the training set involves manually labeling selected objects in the images as synapses or non-synapses. Then, the program crops out a patch of the image surrounding each object. The size of this patch can be defined by the user in the variable ‘patch\_size’. Currently, it is set at 125 x 125 pixels. Following this, while building features during the analysis, each object is rotated and cropped down further defined by the variable ‘crop\_down’. Currently, this is set to get a final patch of 61 x 61 pixels.

Make sure that both trainingset and testset images are subjected to same treatment,i.e., the parameters are the same.

bw\_thresh = 0.10; % lower value -> keep less.

min\_synapse\_size = 300; % min # of segment pixels for synapses.

max\_synapse\_size = 60\*60; % max # of segment pixels for synapses.

min\_synapse\_perimeter = 90; % min # of segment pixels for synapse perimeter.

patch\_size = 125; % size of the patch taken around the centroid. previously was 75

**Setting up the required packages**

1. Designate a root folder. In this example, it is *E:\EMImages*.
2. Download and copy the *Tools* folder into the parent folder, as in *E:\EMImages\Tools*.

Alternatively, download and copy the files from the websites listed below into the *Tools* folder.

* 1. The ba\_interpolation package, available here -   
     <http://www.mathworks.com/matlabcentral/fileexchange/20342-image-interpolation-bainterp2>
  2. MR8 filter bank, available here -  
     <http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html>
  3. HoG descriptors, available here -   
     <http://www.mathworks.com/matlabcentral/fileexchange/33863-histograms-of-oriented-gradients>
  4. LibSVM, available here -  <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
  5. the RFS filter - <http://www.robots.ox.ac.uk/~vgg/research/texclass/>

1. Run the script *load\_packages.m*. This will compile all the required MEX files.

(Type ‘load\_packages’ in the command window and hit ‘Enter’).

**Analysis of images for synapse detection**

1. **Folder structure –**

All images from a single sample is stored in a specific folder. The folders of different samples are in one parent directory which resides in the root folder (here – E:\EMImages) where all the packages have been installed.

1. **Labeling the synapses**

This step lets the user manually label synapses across all samples and save the labeled images in a single folder

Program to be used – *build\_training\_data\_from\_segments.m*

While calling the function,

1. Specify the parent folder containing all samples.
2. Name a new folder where the program will store all the labeled objects, e.g. ‘trainingset\_P1839’.

It should be called like this –

>> build\_training\_data\_from\_segments(‘P-1839’,'trainingset\_P1839');

1. **Analyzing the images**

Program to be used *run\_synapse\_detection.m*

It should be called like this –

>> run\_synapse\_detection(‘P-1839’,'trainingset\_P1839');

* 1. Parent folder is *P-1839*
  2. The second argument would be the folder you created to store your labeled images (step 2.ii). Here, it would be *trainingset\_P1839*. This is used by *build\_features\_samplewise* to build the feature set for the labeled objects.
  3. The confidence threshold will be determined that will proved a recall of 50%. This is done by generating the ROC curves in the function *determine\_threshold* and then using *get\_threshold* to choose the appropriate threshold.
  4. Then, the function *test\_directory* will go through all samples individually that are part of the parent folder and analyze all the images in each sample folder to detect synapses based on the threshold.

**Patches of synapses**

But yes, unfortunately, the model has to be trained on the exact same patch sizes, so you can't just apply the same model to a bigger patch. This means you should:  
1) change the two sets of numbers before you collect training data.  
one's in process\_patch as i mentioned, and the other is in  
build\_training\_data\_from\_segments i think. the latter currently stores  
the patches as 75x75 right now in .mat files so if you want 100x100,  
change it here.  
3) use the training data to build the model  
4) change the numbers in the test\_directory file, as well so when you  
apply the model, it'll be on patches of the same size.

**Building your own Model:** In case you want to build your own Model from scratch do the following

In Matlab,

>> [Features,Labels,Mapping] = build\_features\_samplewise(Dir,croppix)  
>> libsvmwrite('Examples.txt',Labels,sparse(Features));

(croppix = 33 for 125x125px trainingset image -> 61x61px after rotation to align; croppix = 13 for 125x125 -> 101x101; croppix = 8 for 76x76 -> 61x61)

Then in python do a grid search,

$ ./grid.py Examples.txt

This will take a while, maybe 24 hours, depending how fast your computer is and how many examples you have. It will output the values of the two parameters ('c' and 'g'). You can read more about what  
these parameters mean here: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>  
  
Now that you have 'c' and 'g', go back to MATLAB and train the classifier using these values as shown below,  
  
>> Model = svmtrain(Labels,Features,'-b 1 -c 32.0 -g 0.125 -q');