This list serves as an initial outline for a more comprehensive documentation of the workflow for the emdrp pipeline. The starting point is aligned raw data in Knossos format (the alignment and writing of data in Knossos format is done exclusively by Kevin).

See also EM\_pipeline\_time\_mem\_storage.xls for details of run time, memory and storage requirements and also abbreviated version of workflow dependencies (schematic).

30 March 2017, Paul Watkins

1. Delete any unnecessary files and old runs off of NAS, biowulf and gpu clones (particularly biowulf as space is limited by quota and gpu clones as this is more of a caching area since local /Data is needed for streaming out knosos-style cubes of probs)
2. create raw hdf5 from Knossos cubes
   1. make\_hdf5\_from\_knossos\_raw.m run on blue
   2. copy raw hdf5 to nas
   3. copy ds raw to biowulf (if needed)
3. create downsampled version in xy to get closer to isotropic voxels
   1. dpResample.py run on blue
   2. copy ds raw dhf5 to nas
   3. copy ds raw to biowulf
4. manually label at least 3 representative volumes of size 256x256x32 if not using ortho networks OR 128x128x128 if using ortho networks, avoid edges because networks need context outside labeled area
   1. for watkinspv, 3 volumes takes about 40 hours of full time work, varying depending on the complexity of the neurites being traced (areas with large dendrites and cell bodies go much faster)
   2. label ECS with label 1 and label all other neurites / cell bodies with unique label ids, trace up to but not including the membrane that defines boundaries
   3. decide on smoothing and minsize parameters and run label\_checker.sh (dpCleanLabels.py steps) on blue
   4. create label hdf5 label\_maker.sh run on blue
   5. copy label hdf5 to nas
   6. create a cropped version of raw data that only contains the labeled areas (plus 1 knossos cube of context in each direction), can be used for training (convnets and supervoxel classifier), data\_cropper.sh
5. train convnets on all training data
   1. setup and test new .ini / architecture combo and test on blue, need to either sample or comprehensively iterate over some portion of dataset to estimate overall mean and std used by preprocessing in parseEMdata.py
   2. decide on combination of networks (typically 4)
   3. train MORE networks than needed incase of gpu errors or changing mind about using ortho networks, etc
   4. train networks on gpu clones
   5. copy trained convnets to all clones (so far found that best division of labor is each gpu clone processes 1/number\_of\_clones of the dataset, and each of 4 gpus on each clone uses a different trainined network)
6. train convnets using x-fold cross validation (needed for agglomeration training)
   1. use same number of networks per cross validation
   2. may have to modify x-validation if more than about 8-10 training volumes (either by sampling of leave out one, or some other leave out combinations)
7. export training volumes for cross validation (using shell script, aka run\_emneon\_batch\_xfold\_out.sh)
8. in parallel:
   1. copy trained convets to all clones
      1. pick superchunk size, typically 8x8x4 or 6x6x6, try to make close to square depending on anisotropy
      2. pick overall output volume size so that it’s multiple of superchunk size pick and also of number of machines, include a cube of boundary that is additionally exported for each network so overlap can be included
      3. export trained convnets over all of dataset or some pre-defined portion of dataset, export in Knossos format (data\_cubes)
      4. NOTE: make sure not to exceed disk space on clones, might have to use multiple iterations to export whole area, in which case move to step 8
   2. copy xfold probs to red
      1. run watershed at (xxx – how to decide on thresholds) for test volumes
      2. train supervoxel classifier on test volumes
      3. copy trained classifier dills to biowulf
9. merge probs from different networks simultaneous with copy to NAS (i.e., “push” method for prob aggregations)
   1. do this serially per gpu clone machines but write out per superchunk
   2. use filemodulator\_overlap mode of dpCubeIter to minimize writes for overlap
   3. delete cubes off of clones after completion
10. copy probs from nas to biowulf
    1. separate into n folders (xxx – copy command here) and use bbcp or rsync WITH –e ssh switch from each of n machines, which currently are green, red, infra (vm on green) and ultra (vm on green)
    2. for prob files currently saturate disk rate (~140 MB/s with WD NAS 8TB drives, RAID5) at only 2 machines copying simulataneously
11. run watershed against probs on biowulf in parallel for each “superchunk” volume on biowulf
    1. use dpCubeIter to generate inputs to swarm
12. run agglomeration on biowulf
    1. setup “parallel” config file
    2. use dpCubeIter to generate inputs to swarm
13. run “cleaning” on biowulf
    1. first do “clean write” which copies voxel type to agglo hdf5s
    2. cleaning removes small supervoxels and merges them with nearest neighbors and removes cavities, then recomputes supervoxel types and re-orders labels (for ICS then ECS and updates types\_nabels)
    3. should run in less than 30 mins per superchunk, so run on quick nodes
    4. usually only clean labels from whatever agglomeration iteration that is going to be used, otherwise for multiple iterations, setup dependencies to avoid multiple hdf5 writers issue
14. copy cleaned labels from biowulf back to synology
    1. depending on size might benefit from up to 4 machines (as smaller files transfer quickly and before line/disk caches are saturated)
15. export data to CDCU fileshare
    1. raw data also if using downsampled
    2. can export probs by converting to uint8 grayscale
    3. make sure to enable compression for segmentation labels
    4. xxx – modified Knossos that can support multiple label sets?