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).

Started a folder for each dataset under emdrp repository at scripts/shell/<dataset\_name>. References to shell scripts below will be in this directory. This serves as a preliminary documentation for all backend parameters used for a particular run (along with github release tag, should be sufficient for reproducibility).

26 June 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. automatically export volumes based on locations picked from browsing through dataset using Knossos (label\_cube\_gen.sh)
   4. decide on smoothing and minsize parameters and run label\_checker.sh (dpCleanLabels.py steps) on blue
   5. create label hdf5 label\_maker.sh run on blue
   6. copy label hdf5 to nas
   7. 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), particularly ideal for classifier on machines where entire dataset copy is not necessary, like training supervoxel classifier on red (data\_cropper.sh)
5. Typically current methodology is to train everything on each gpu clone (5x) with all possibilities needed for all training data and leave-out-one for agglo training run on each gpu clone. Then test for any failed jobs and discard these, or if all jobs finish without error, then just leave one of the clones out for averaging the probs. NOTE: still use all clones for exporting the training volumes (next step).
   1. 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 (emneon\_train.sh)
      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)
   2. train convnets using x-fold cross validation (needed for agglomeration training), (currently also emneon\_train.sh)
      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)
6. currently somewhat hacky method of renaming / moving convnet outputs, typically written to ~/Data/convnet\_out, created script (mega\_mover.py) for programmatically figuring out proper renames based on text output and ini files, copy this to convnet output location and modify appropriately
7. export training volumes for cross validation (emneon\_export\_xfold.sh), export with context if using the context mode of the supervoxel classifier
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 with one cube of context in each direction if intending to stitch with overlap and/or intending to use context mode of supervoxel classifier
      4. export trained convnets over all of dataset or some pre-defined portion of dataset, export in Knossos format (data\_cubes)
      5. 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
      6. NOTE: alternatively copy trained convnets to one of the synology so they are accessible to all clones
   2. copy xfold probs to red
      1. merge probs together and run watersheds for each test chunk (merge\_watershed\_xfold\_K0057.sh)
      2. NOTE: if the test probs were output separately during xfold export, then the order that the chunks are processed in the shell script must match that from the EM parser .ini file for the convnets
      3. train supervoxel classifier on test volumes (classifier\_train\_xfold.sh)
      4. 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 (aggregate\_export\_probs.sh)
   2. most likely these probs will not be saved permanently, so stream to both synologies so that bandwidth can be maximized
   3. use filemodulator\_overlap mode of dpCubeIter to minimize writes for overlap, NOTE: volume\_range\_beg and \_end should still included overlap volumes, this is handled by overlap flag to dpAggProbs.py
   4. delete cubes off of clones after completion
10. copy probs from nas to biowulf
    1. separate into n folders (split\_transfer.sh) 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. xxx – next time test speed with bbcp
11. watershed against probs on biowulf in parallel for each “superchunk” volume on biowulf
    1. use dpCubeIter to generate inputs to swarm (volume\_watershed.sh)
    2. NOTE: must use lscratch as IO errors (particularly write errors for hdf5 outputs) are very common either resulting in “stale NFS file handle” or completely hanging some jobs with no error reported
12. run agglomeration on biowulf
    1. setup “parallel” config file
    2. use dpCubeIter to generate inputs to swarm (volume\_agglomeration.sh)
    3. NOTE: must use lscratch as IO errors (particularly write errors for hdf5 outputs) are very common either resulting in “stale NFS file handle” or completely hanging some jobs with no error reported
13. run “cleaning” on biowulf
    1. first do “clean write” which copies voxel type to agglo hdf5s (volume\_copy\_type.sh, script automates swarm submission using dependencies to avoid multiple writers issue)
    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 (volume\_clean.sh)
    5. added “multiple” segmentation feature to Knossos to be used to provide different levels of segmentation incase of merger. Export to different datasets within one hdf5 (using job dependencies) or different hdf5s
    6. might want to compare against cleaned watershed outputs also (volume\_clean\_watershed.sh)
14. copy cleaned labels from biowulf back to synology
    1. use at least 4 machines for copy, as with probabilities, copy to synology with RAID5, as this is final output, so better to have redundancy
    2. may also want to copy agglo or watershed outputs back to one of the synologies as backup and for when they have to be deleted off of biowulf
15. meshing, xxx – fill me in, also discuss multiple levels feature (definitely want different hdf5 files in this case b/c of multiple writers and long runtime for large supervoxels)
16. export data to CDCU fileshare
    1. segmentation data, make sure compression is enable (xxx – shell script, currently on green)
    2. raw data also if using downsampled
    3. can export probs by converting to uint8 grayscale (xxx – ability to do this using the new static dataset feature from rutuja, written to same directory structure)
    4. xxx – modified Knossos that can support multiple label sets?