Mounting krs0014 hard drive disk for Ubuntu with Read/Write permissions.

http://tuxtrix.com/2014/03/accessing-hfs-partition-from-linux-in.html

krs0014 HDD (Asset tag : KRS 0014) has been journaled under Mac OS. Permissions now have to be explicitly forced when mounting on an Ubuntu/Linux distribution. All the following commands have to be run as root using ***sudo.***

First check your disk to find the HFS/HFS+ partition that corresponds to krs0014 (Here it's /dev/sdb1/) :

```
 sudo fdisk -l
```

```
**   Device Boot       Start        End          Blocks       Id    System
**/dev/sdb1   *          63   3906959804  1953479871  af   HFS / HFS+
```

Unmount HDD if it has been automatically mounted. (You can check it by running “mount” with no arguments).

```
 sudo umount /media/${user}/krs0014
```

Create mount point

```
 sudo mkdir /media/${user}/krs0014
```

Run :

```
 sudo mount -t hfsplus -o force,rw /dev/sdb1 /media/${user}/krs0014
```

If no warning message is shown, then check that /dev/sdb1/ has the “rw, force” tags when running “mount” with no arguments. If so, you are all set.

Otherwise, if you get : “`mount: warning:  /media/${user}/krs0014 seems to be mounted read-only.`”, then run :

```
 sudo fsck.hfsplus -f /dev/sdb1
```

The output should be the following :

```
** /dev/sdb1
** Checking HFS Plus volume.
** Checking Extents Overflow file.
** Checking Catalog file.
** Checking multi-linked files.
** Checking Catalog hierarchy.
** Checking Extended Attributes file.
** Checking volume bitmap.
** Checking volume information.
** The volume /media/${user}/krs0014 appears to be OK.
```

Finish by re-running :

```
  sudo umount /media/${user}/krs0014
  sudo mount -t hfsplus -o force,rw /dev/sdb1 /media/${user}/krs0014
```

The global aim of this project is to extract blood vessels from ultrasound scan, in order to detect the formation of micro-tumors, based on tubes tortuosity measures.
The approach is using convolutional neural network as a pixel classifier. Every input to the network corresponds to a 65x65 patch extracted from the input image. It outputs the probability for the central pixel to be a blood vessel.

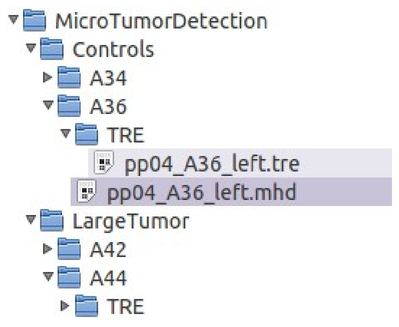
Prerequisite :
● Installation
- Download Caffe from https://github.com/BVLC/caffe and install following the build instruction here :
http://caffe.berkeleyvision.org/install_apt.html
http://caffe.berkeleyvision.org/installation.html#compilation
- Install Cuda from the Nvidia website https://developer.nvidia.com/cuda-downloads and then build Caffe by commenting line 8 “CPU_ONLY := 1” and setting “CUDA_DIR := /usr/local/cuda-7.5” in Makefile.config.
- Build pycaffe.

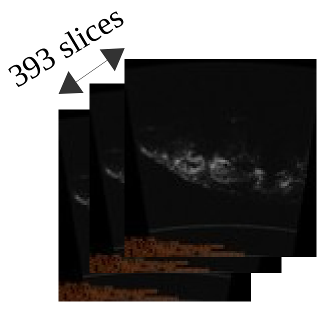
● Data structure
- Please keep in mind that all scripts have to be run from the **caffe build directory root**.
- Path to files are hardcoded in the scripts and should be given as arguments. Be sure that all directories exist before running the script.
- The outpoint point map used in 5-b are generated in 1-b for every input data. In order to save disk space, we could generate those points only for the testing cases.

1. Input data structure and conversion

a) Structure

Input data are raw ultrasound scans (.mhd MetaImages) associated with their expert vessel segmentation (.tre files).
They are divided in two folders in order to distinguish the Controls files from the LargeTumors ones. Scripts are actually looking for those files in *${caffe_root}/data/MicroTumorDetection/Controls/* and *${caffe_root}/data/MicroTumorDetection/LargeTumor/*. Where ${caffe_root} is the path to your caffe build.
Animals are separated according to their name (A34, A36...) inside of those directory. Each animal directory contains a “TRE/” directory populated with the animal expert segmentation files.





At this time, 20 control images and 20 tumor images are used as input data. More input data are stored on krsdata1 and can be accessed at “smb://krsdata1/data-notbackedup/Local/Lucas/TumorMicroEnv/”
Keep in mind that later, every input image will produce an average of 12000 training input images for the neural network.

b) Conversion

Script: **ConvertData.py**
Aim: **Shrink image into 10 slabs**

We cannot use the raw input directly because of the lack of information in each slice. Each animal scan will be shrunk into 10 slabs along the z direction. Each slab contains the Max Intensity Projection of pixels within its requested region, giving a nice enhancement of vessel pixels coming from this area.

Expert files are first converted to .mha MetaImages and then shrunk the same way as input scans in order to map the expert extraction with the input ultrasound scan.

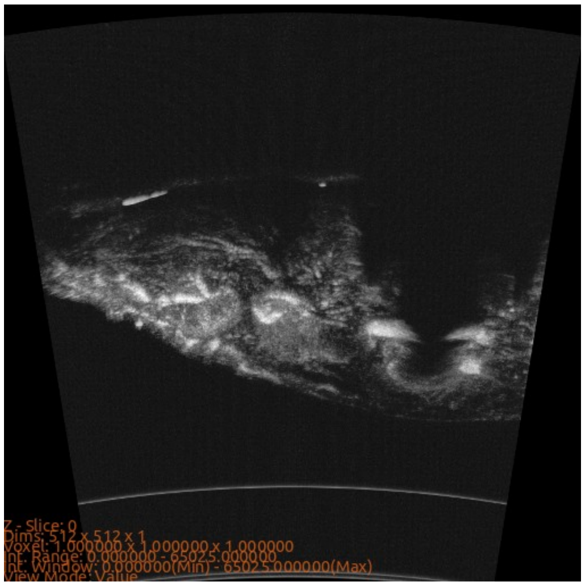
This is done by the ConvertData.py script. It uses TubeTk's CLI ShrinkImage and ConvertTubesToImage to process the previously described actions.
Finally, the script copies the output slabs into our testing and training directories that will be used by the network. Control files and Tumor files are mixed to constitute generic training and testing sets.

At this point, we are working using an external hard drive in order to store the huge amount of data we are about to deal with. Testing and training input images are saved here *${hardDrive_root}/MicroTumorDetection/training/* and *${hardDrive_root}/MicroTumorDetection/testing/*
Both directories are storing input files and expert files respectively in the *images*/ and *expert*/ sub-directories. They also contain an "*out*/" sub-directory to save the 65x65 patches that will be used later to create the network's training and testing sets, and a "points/" sub-directory to store the 3D point map that will be used to map the slabs back to 3D image space..

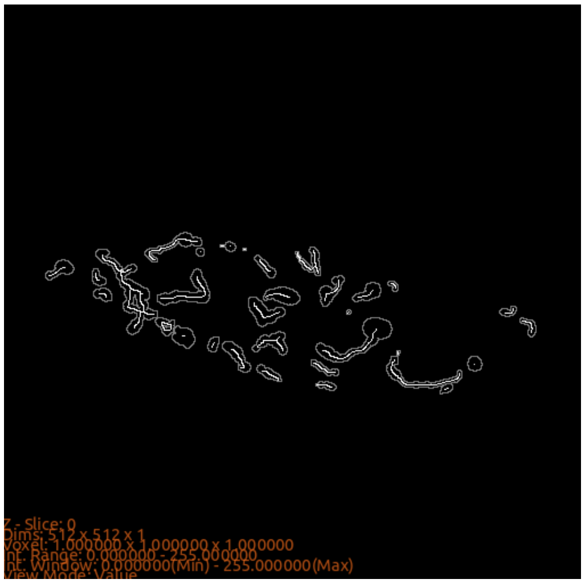
c) Compute training mask

Script: **saveSlabs.py**
Aim: **Save slabs and compute expert training mask.**
ITK python wrapping is required

We have to save each slab separately as we cannot deal with 3D images. This is done by prepending the slab id to the actual filename.
Before saving the expert images, we compute them into a label map.
Each central pixel will be used as positive input and will generate a 65x65 patch. For each of those patches, the corresponding network output should be 1.
Pixels located on the vessels out layer are used as negatives and correspond to the network output 0.



```
Z - Slice: 0
Dims: 512 x 512 x 1
Voxel: 1.000000 x 1.000000 x 1.000000
Int. Range: 0.000000 - 65025.000000
Int. Window: 0.000000(Min) - 65025.000000(Max)
View Mode: Value
```



```
Z - Slice: 0
Dims: 512 x 512 x 1
Voxel: 1.000000 x 1.000000 x 1.000000
Int. Range: 0.000000 - 255.000000
Int. Window: 0.000000(Min) - 255.000000(Max)
View Mode: Value
```

2. Create training and testing set.

a) Create patches

Script : **PreProcessing.py**
Aim: **Create 65x65 training patches and write the corresponding output to a text file.**

At this point we have 200 input slabs with their 200 associated label maps in each training and testing directories.
We iterate over the expert label map and create a 65x65 patch around the current pixel if its label is positive. The same amount of negative patches are randomly picked. 80% of them are coming from the vessel bounds, 20% from the entire image. This allows enhancement in vessel boundaries detection but reduces background denoising.
Patches are save int the “out/” subdirectory of each training and testing directories.
For every patch, the expected network's output and its associated filename are written to a .txt file in order to create the database.
Text files for training and validation are currently located at :
${caffe_root}/data/MicroTumorDetection/train.txt
${caffe_root}/data/MicroTumorDetection/val.txt

b) Create LMDB data base

Script: **create_imageset.sh**
Aim: **Create Caffe database for training and validation.**

We use caffe create_imageset.sh script to create an lmdb database from our 65x65 patches.
It's looking for input files in */media/${user}/krs0014/MicroTumorDetection/training/out/* and will create the database in *${caffe_root}/data/MicroTumorDetection/Net_TrainData*. Identically, it will create Net_ValData using the testing directory.

Note that both *Net_TrainData* and *Net_ValData* directories should not exist when you run the script.

3. Neural Network

a) Training

Script: **TrainNet.py**

This script uses pycaffe to train the network using our database. Caffe uses protofiles to save the network architecture and weights once it is trained. Protofiles are saved under ${caffe_root}/data/MicroTumorDetection/NetProto/
The same files will be used later to process the network.

The path to those files and all network's architecture and training parameters are accessible through the script.
Additionally, utility functions are provided to visualize features, outputs and performances.

b) Process the network

Script: **ProcessNet.py**

This script takes an animal name as input and successively send all the corresponding slabs as an input to the network. The network status is read from the protofiles.
Each pixel of the current input slab is classified using a 65x65 patch after its index has been saved.

We then map the network output value back to the saved index in order to reconstruct the slab.

5. Reconstruct image

a)Reconstruct slabs

Script: reconstructSlabs.py
Aim: Save all processed slabs into a single image for the specified animal.
ITK python wrapping is required.

This script uses the animal name to retrieve all the corresponding slabs that have been processed and save them as a single .mha image.

b) UnShrink

Script:UnShrink.py
Aim: Map each pixel back into 3D space.
ITK Python wrapping is required.

This script uses the output point map from the ShrinkImage CLI to map the slabs pixel back into the original image 3D space.

6. Extraction

Script: ExtractVessels.py
Aim: Extract vessels using SegmentTube CLI

We extract vessels from the raw input image using the 3D reconstructed image as vessels seeds.
The .mtp vascular model has been computed from a previous extraction using EnhancetubeUsingDiscriminantAnalysis CLI.
The scale image can also be obtained that way, or can be replaced by a constant.