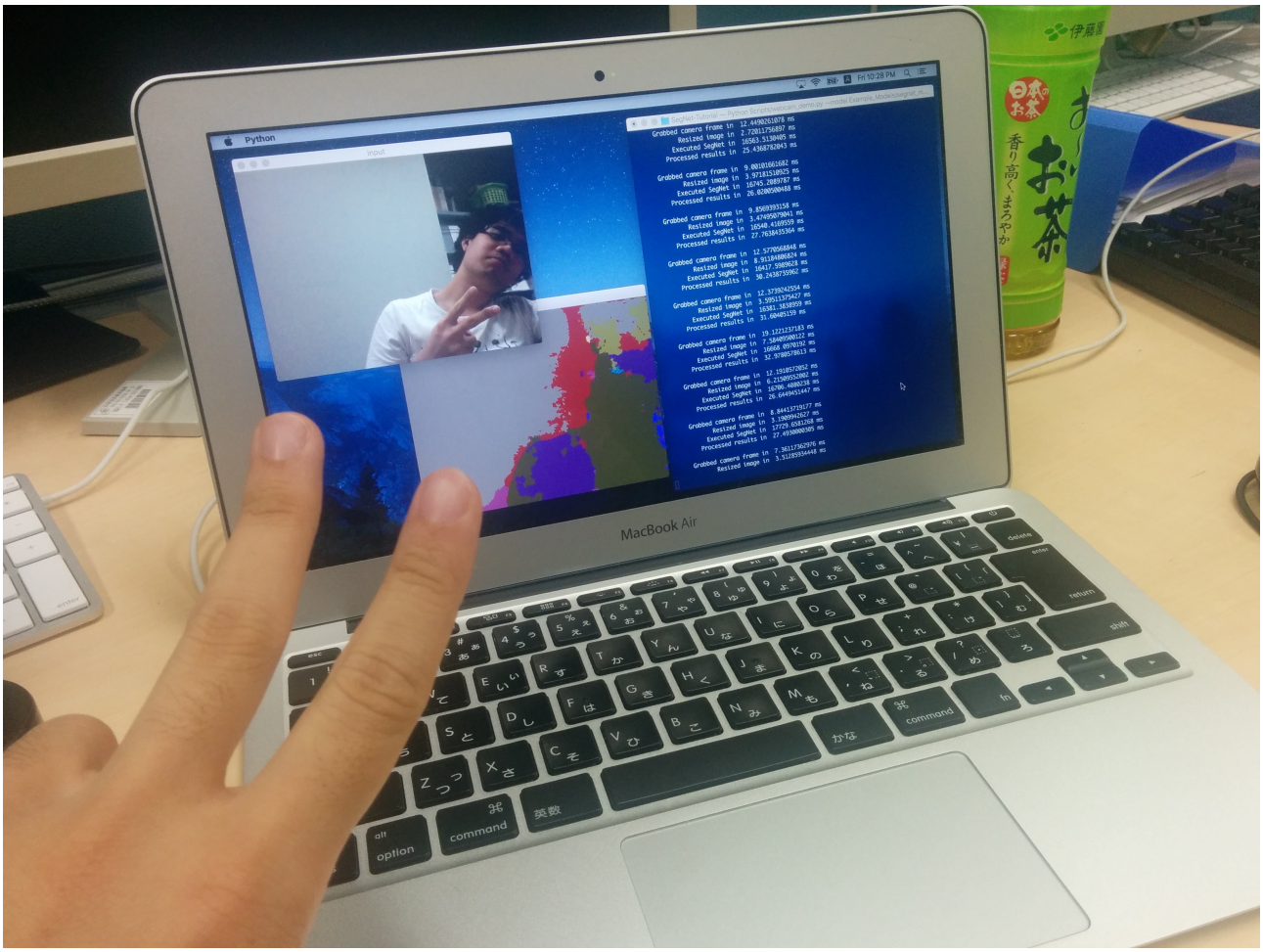
How to set Caffe up and running SegNet from scratch in OSX El Capitan using CPU mode



Written by : Sirawat Pitaksarit

Probably can be applied to Ubuntu system, but all the brew commands needs to be replaced by apt-get which the package name might not be exactly the same.

Machine

MacBook Air (11 inch, Mid 2011) will be used. This machine has no GPU so the installation will include only CPU support. This guide starts from clean install of El Capitan and will takes about 2-3 hours.

Install Homebrew

For easy installing of everything else.

Press Ctrl+Space and type Terminal then paste this command

```
  /usr/bin/ruby -e "$(curl -fsSL https://
raw.githubusercontent.com/Homebrew/install/master/install)"
```

This will also download Command Line Tools for Xcode. (contains git, etc.)

Clone Caffe project

```
  git clone https://github.com/BVLC/caffe.git
```

Install dependencies

General dependencies

```
  brew install -vd snappy leveldb gflags glog szip lmdb
  brew tap homebrew/science
  brew install hdf5 opencv
```

(About 3 minutes)

Protobuf and Boost

Protobuf is from Google. It provide a main data communication mechanism in Caffe.
Boost is a general multipurpose package for python, c++

```
  brew install --build-from-source --with-python -vd protobuf
  brew install --build-from-source -vd boost boost-python
```

(Protobuf takes about 2 minute, Boost takes about **30 minutes** so hit CMD+N to open new Terminal and continue to next step while Boost compiles.)

Modifying Protobuf from source

The protobuf from previous section has one problem : if the protobuf message is too large the program will refuse to work for security reason. Normal Protobuf usage will not exceed the limit but in Neural Network (especially with image) it is very easy to exceed this limit. We have to edit Protobuf message limit from source and compile + install maually. For more information see http://stackoverflow.com/a/35172491/862147 The correct solution is not to edit this limit but to modify the

program to use smaller message, but in many case modifying program is not possible so editing the limit is required.

And also Protobuf from previous command is old. (2.6) This step will get you Protobuf 3.0

Get Protobuf project from GitHub.

```
git clone https://github.com/google/protobuf.git
cd protobuf
vim src/google/protobuf/io/coded_stream.h
```

We will edit this coded_stream.h file with vim. You can use any other editor if you like.

```
/kDefault (search for the variable)
ww (go to next 2 words)
cw512[ESC] (limit is now 512 MB instead of 64MB)
n (find next)
ww (go to next 2 words)
cw256[ESC] (warning threshold is now 256 MB instead of 32MB)
:wq (save and quit)
```

Next we will compile Protobuf. But we have to install the compilation tools first

```
brew install autoconf automake libtool curl
```

Compile Protobuf for C++, now we are still in protobuf folder :

```
./autogen.sh
./configure
make -j4
make -j4 check
sudo make install
```

make is about 5 minutes. make check another 5 minutes. (Test should says “PASS : 6”)
Check the installation

```
protoc --version
```

It should says “libprotoc 3.0.0” instead of “libprotoc 2.6.1”

Next, install Protobuf for python.
First we will install a new python to replace the system’s python and then install Protobuf for python. Lastly we run sudo make install again to cancel out the unlink.

```
cd python
brew install python
brew unlink protobuf
python setup.py clean
python setup.py build
python setup.py test
sudo python setup.py install
cd ..
sudo make install
```

Install BLAS : OpenBLAS

BLAS is a library for doing linear algebra. Caffe can use : ATLAS, OpenBLAS or Intel MKL. In El Capitan there is an ATLAS built in but came with some problem. (Cannot include <cblas.h>) So we will install OpenBLAS with brew and use it instead of ATLAS.

```
brew install homebrew/science/openblas
```

Compile Caffe

Finally we can compile Caffe. We have create **Makefile.config** file. In here we can edit some options. For Macbook Air we must enable CPU mode.

```
cd caffe
cp Makefile.config.example Makefile.config
vim Makefile.config
```

Then uncomment the CPU_ONLY line.

```
CPU_ONLY := 1
```

And on this line

```
BLAS := atlas
```

Change to

```
BLAS := open
```

To use OpenBLAS instead of ATLAS. And a little bit below that **uncomment** this 2 line

```
BLAS_INCLUDE := $(shell brew --prefix openblas)/include
BLAS_LIB := $(shell brew --prefix openblas)/lib
```

To use the OpenBLAS that brew just installed in previous step.

Our python headers is not in the directory that Caffe says because our python came from brew, so change

```
PYTHON_INCLUDE := /usr/include/python2.7 \
   /usr/lib/python2.7/dist-packages/numpy/core/include
```

to

```
PYTHON_INCLUDE := /usr/include/python2.7 \
  /usr/local/lib/python2.7/site-packages/numpy/core/include
```

Our python lib is also not in the correct brew path, so change

```
PYTHON_LIB := /usr/lib
```

to

```
  PYTHON_LIB := /usr/local/Cellar/python/2.7.10_2/Frameworks/
Python.framework/Versions/2.7/lib/
```

Warning! On the path that contains version number, in your machine might not be the same as mine so please check the correct folder name.

After saving the Makefile.config, we can build Caffe. Note that **-j4** assume that you have 4 cores CPU. If you have more you can increase this number. (Easily determine cores with **htop** command which you can get using brew. After running **htop** you can see on the top how many bars of CPU activity you have.)

```
  make -j4
```

Takes about 20 minutes, after that we will test the compilation :

```
  make -j4 test
  make -j4 runtest
```

The first command make the test, the second command run the test we just made. Takes about 10 minutes. If it says PASSED the Caffe is ready for use!

Next, we will install **python interface** too.
First we will install python dependencies. Now we are still in caffe main folder.

```
  for req in $(cat python/requirements.txt); do pip install $req;
done
```

And this fix a problem in OSX that could occur when running pytest. (“unknown locale: UTF-8”)
Create ~/.bash_profile

```
  vim ~/.bash_profile
```

Type the following environment variable addition command in the file (press i to enter insert mode, after you done press ESC) and then save + exit (:wq).

```
  export LC_ALL=en_US.UTF-8
  export LANG=en_US.UTF-8
```

Reload the .bash_profile file by using

```
  source ~/.bash_profile
```

Next, make python interface.

```
  make -j4 pycaffe
  make -j4 pytest
```

(Do not use sudo while running python test, it might result in libcudart.so not found)

Check that Caffe is ready in python by using iPython. Currently we are in caffe main folder. Python interface is in the python folder.

```
  cd python
```

```
pip install ipython
ipython
```

In **iPython** try this command.

```
In [1]: import caffe
In [2]:
```

If no error appears then it is ready to be used. Exit iPython with Ctrl+D 2 times.

Note that compiling Caffe don't have installation. It does not install any command to your bash. Everything (both C++ and python) is in this caffe folder.

Test Caffe with SegNet

Caffe comes with some basic neural networks for testing, but we will test with network from other source, SegNet. (https://github.com/alexgkendall/caffe-segnet)

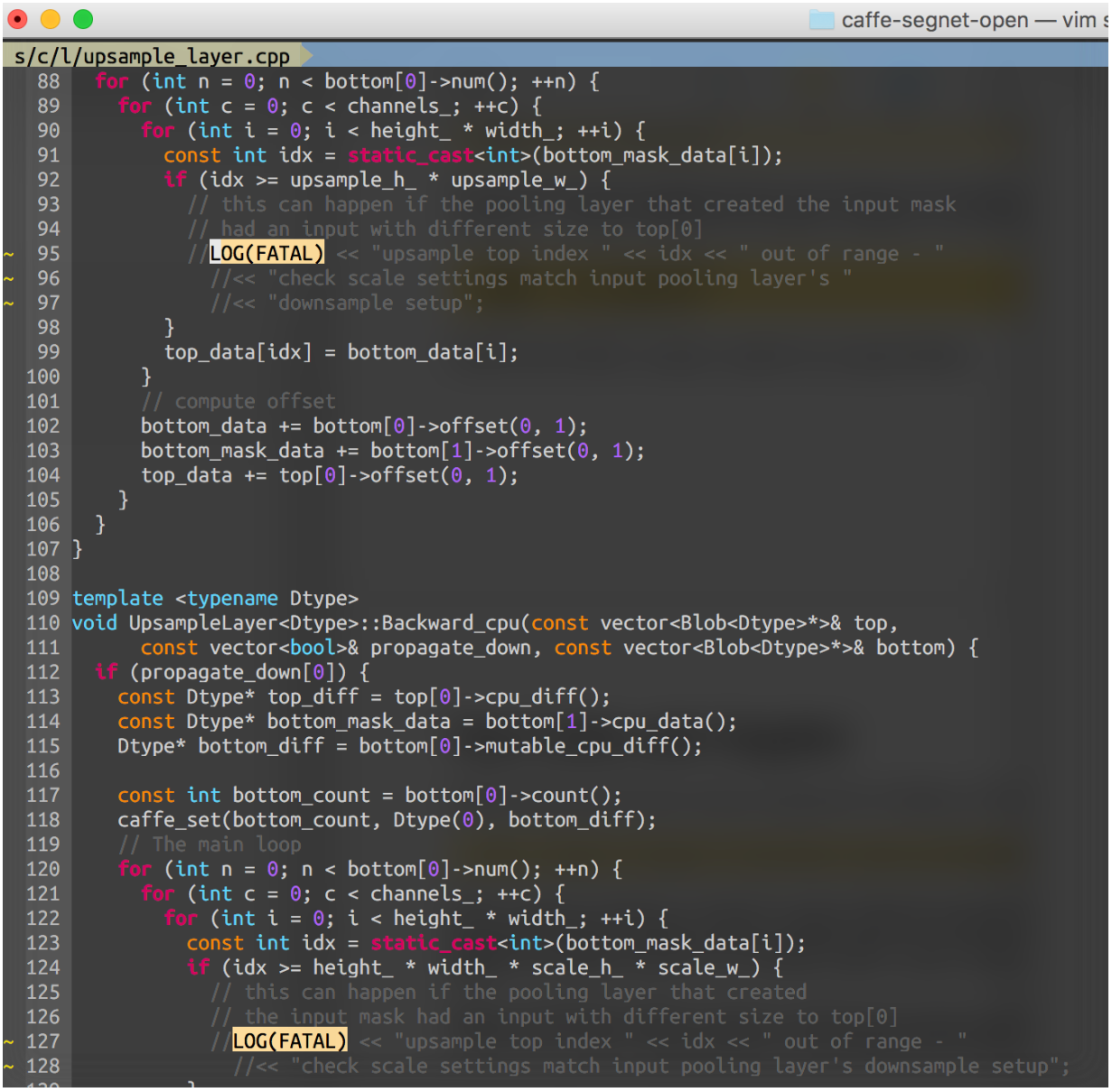
```
git clone https://github.com/alexgkendall/caffe-segnet.git
```

Many network based on research paper **have it's own version of Caffe** inside because they add new layer types to Caffe. So you need to **build caffe again.** Please follow the "Compile Caffe" section again after cd into the cloned repository.

Before compiling Caffe we have to edit some source code. This is the bug with SegNet when using it with CPU mode. (https://github.com/alexgkendall/caffe-segnet/issues/5)

```
vim src/caffe/layers/upsample_layer.cpp
```

Search for LOG(FATAL), there are 2 spot. Comment out these error generation codes.



After this compile the Caffe that comes with SegNet repository by following the same steps.

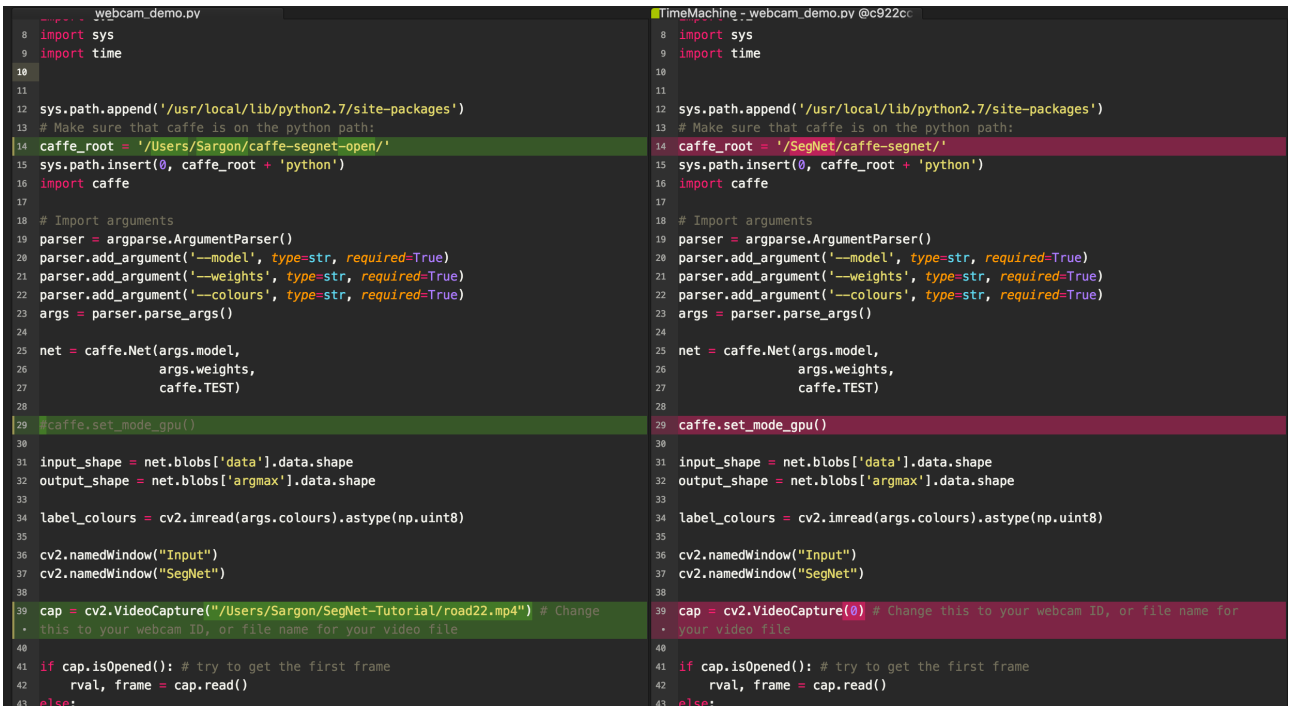
Please note that in the **make -j4 pytest** step it will shows the error “Unknow layer type: Python”. This is because the author of SegNet removed this layer but it is still in the test. Don’t worry about it.

Go back to home folder and clone **another repository** from the same author.

```
cd
git clone https://github.com/alexgkendall/SegNet-Tutorial.git
cd SegNet-Tutorial
```

Edit **Scripts/webcam_demo.py** to specify caffe-segnet folder that we just compiled, comment out the GPU mode and then you can also use video file instead of a webcam. Example below. The path can be relative or absolute, but I used absolute here just to be safe.

For video source, **cv2.VideoCapture(0)** should work by default because Macbook Air have a built in web camera and OpenCV will use it as it is the ID #0 device of this machine.



Next, you have to download pre-trained weight file.

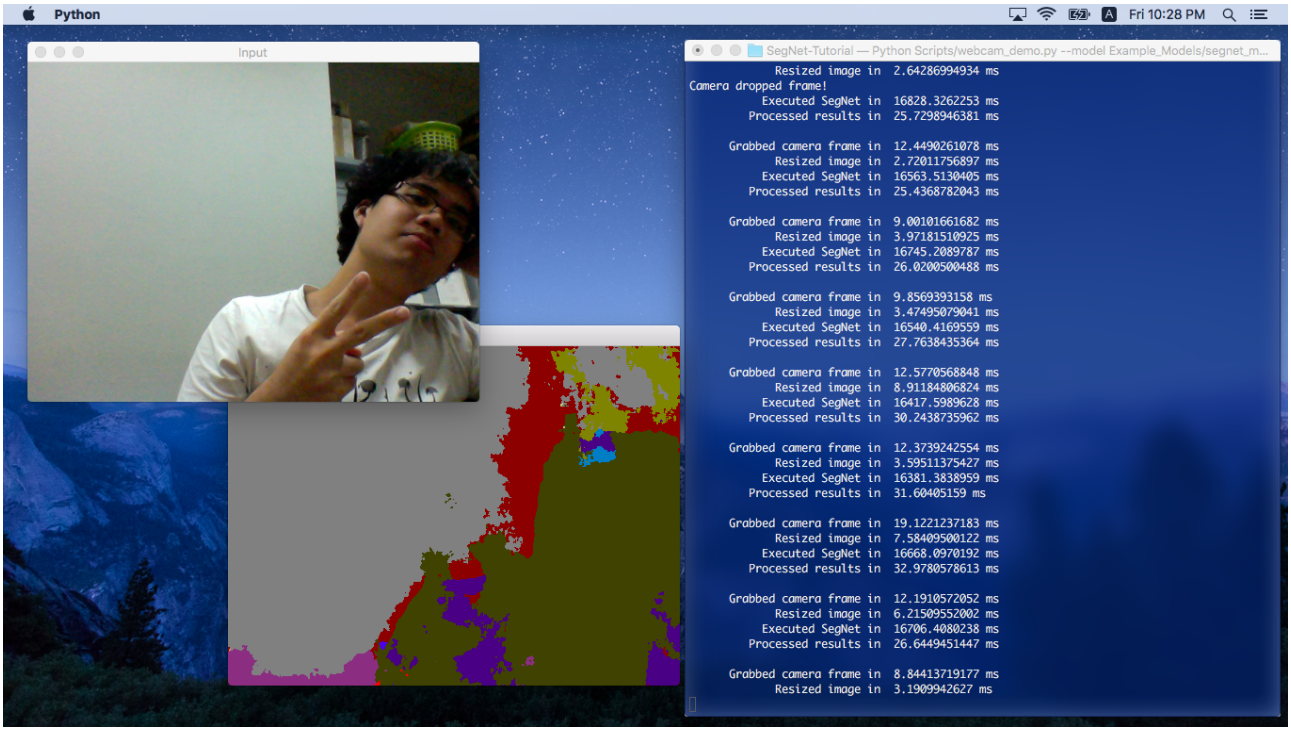
http://mi.eng.cam.ac.uk/%7Eagk34/resources/SegNet/segnet_weights_driving_webdemo.caffemodel

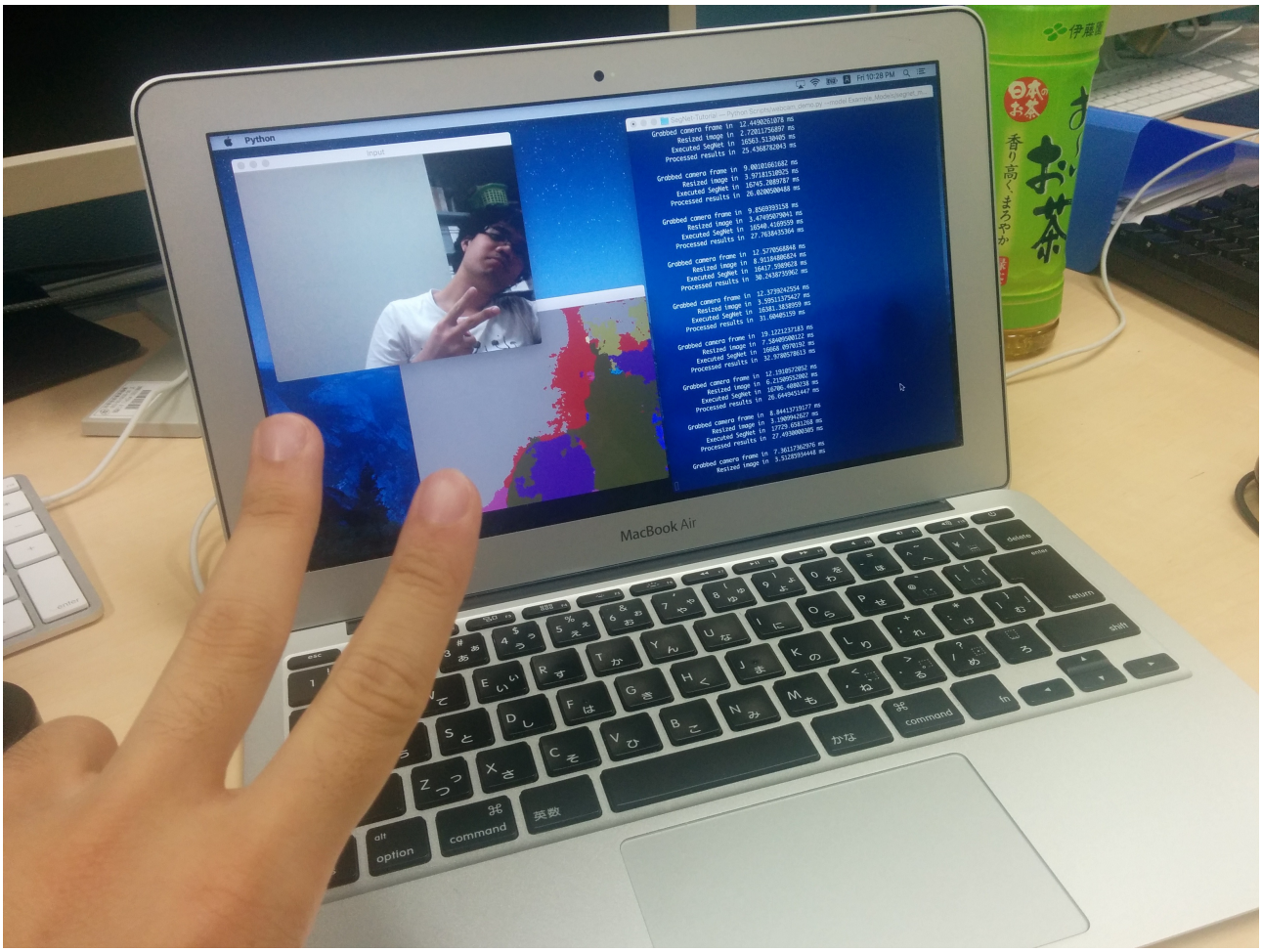
And then move it to the folder inside this project for easy access. This folder already contains corresponding .prototxt file.

```
  mv ~/Downloads/segnet_weights_driving_webdemo.caffemodel Example_Models/
```

Everything is ready. Now run this python command! (Make sure we are in SegNet-Tutorial folder) This command uses model and pre-trained weight, and color code it according to 12 categories defined in the official website.

```
  python Scripts/webcam_demo.py --model Example_Models/segnet_model_driving_webdemo.prototxt --weights Example_Models/segnet_weights_driving_webdemo.caffemodel --colours Scripts/camvid12.png
```





Congratulations! Your Macbook Air is now running Caffe with SegNet! If it works that means both C ++ and python part of Caffe is working fine in CPU mode. Please wait a bit as this is processed by Macbook Air CPU and it can take up to 20 seconds per frame. As for why the output does not look

understandable, it is because this network has been trained to be used on road scene and not indoor scene.

This program is the same as this web demo : http://mi.eng.cam.ac.uk/projects/segnet/ If you want to try the model that will go together well with other .prototxt files in Example_Models/ folder please visit https://github.com/alexgkendall/SegNet-Tutorial/blob/master/Example_Models/segnet_model_zoo.md

Extra : To enable GPU + CUDA + CUDNN capability

GPU has less RAM than CPU, but if you manage to fit your network in the GPU, the computation will be much faster. **Note that this will not work on the MacBook Air (compilation will fail) because this machine has no GPU)**

If you want to modify Caffe's capability, a recompilation is needed. In the Makefile.config these lines has to be modified :

```
USE_CUDNN := 1
# CPU_ONLY := 1
```

Make sure **USE_CUDNN is uncommented** to use CUDNN, library from Nvidia that uses Nvidia GPU to boost neural network speed. Make sure **CPU_ONLY is commented** too so we can use GPU.

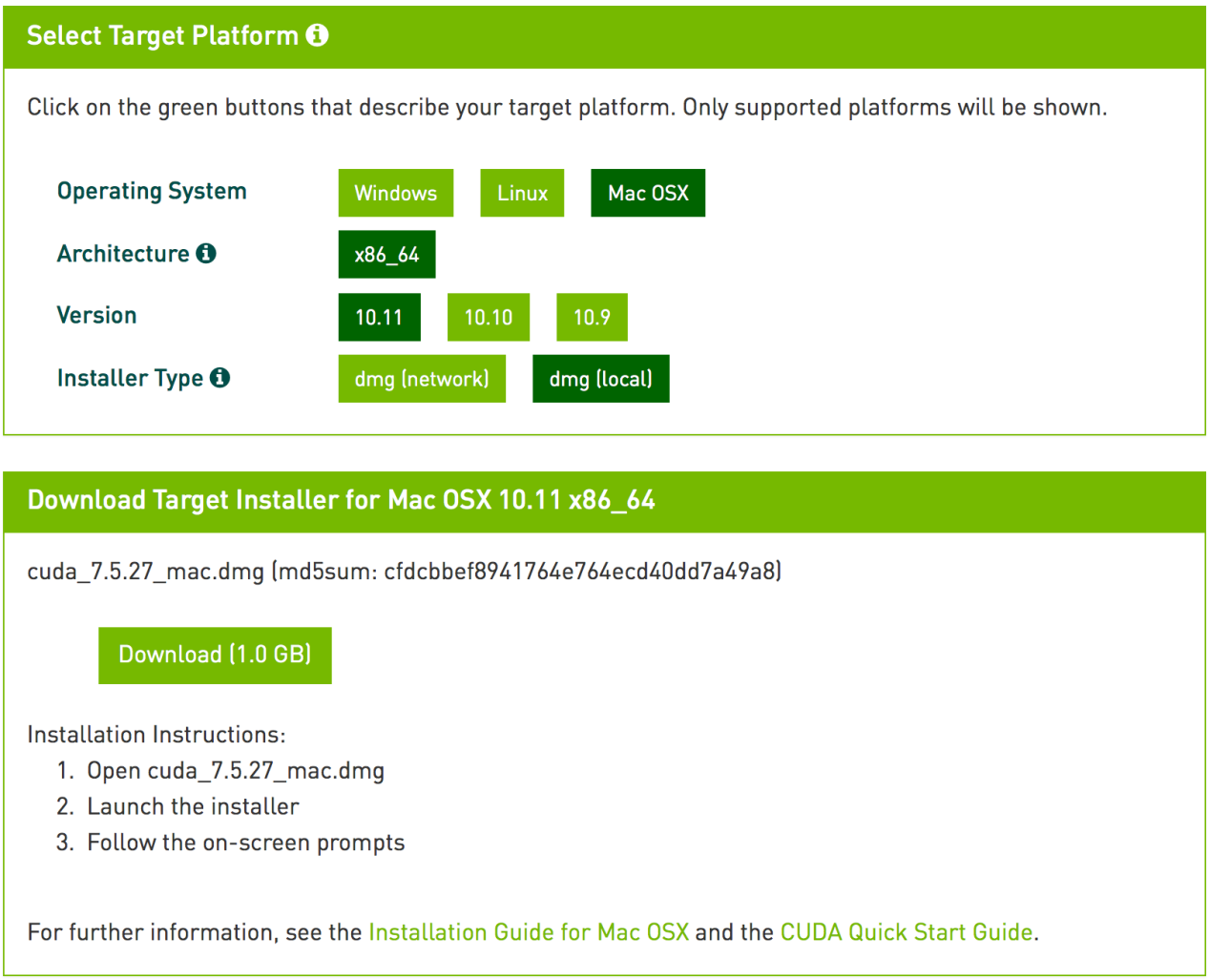
```
TEST_GPUID := 0
```

If you have more than 1 GPU, the GPU ID has to be inputted here so Caffe will use that GPU for testing the compilation. (When actually using Caffe, you can select GPU ID later) If you have 1 GPU, then number 0 is fine.

Note that GPU ID may not be the same as reported from **nvidia-smi** command. In my machine **nvidia-smi** gives #0 Tesla K20c (For computation) and #1 GeForce GTX 960 (Main GPU for displaying and computation also) but enterting ID 0 to caffe actually uses GTX 960.

Install CUDA

Go to this link and download the latest version :
https://developer.nvidia.com/cuda-downloads



Choose the correct driver for your machine, which should not be an OSX because no OSX has Nvidia GPU yet.

After this, **restart the machine.**

```
sudo reboot
```

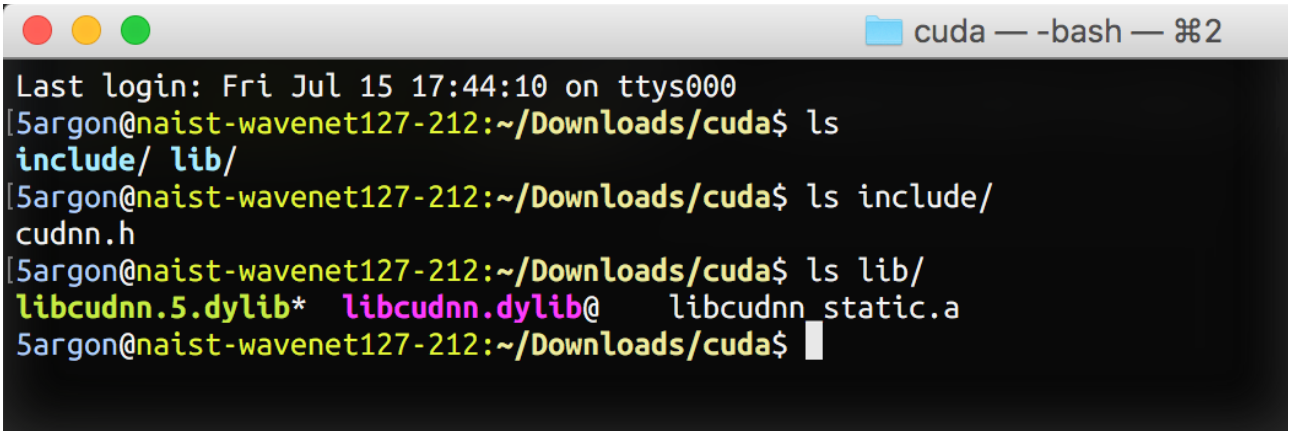
In Ubuntu, installing CUDA is frustrating as it could destroy your graphic card driver. (Like, your graphic cards is too old to use the display driver that came with newer CUDA) Please be prepared if you reboot and cannot get into graphical interface again you have to reinstall the correct Nvidia driver. And then install lower CUDA version. Please check what is the highest version of driver that your graphic card can handle, and then see what is the corresponding CUDA version of that driver.

Getting CUDNN

This requires free membership of Nvidia Accelerated Computing program. Go to this page and register. You have to wait for their confirmation mail (wait for 1-3 days). After that you will be able to download CUDNN for free.

https://developer.nvidia.com/cudnn

The folder (named “cuda”) contains only compiled library and nothing else. To install, you just copy it to correct place. (your “cuda” folder)



To install by copying :

```
sudo cp include/cudnn.h /usr/local/cuda/include
sudo cp lib/libcudnn* /usr/local/cuda/lib
sudo chmod a+r /usr/local/cuda/lib/libcudnn*
```

or if the lib folder is named lib64

```
sudo cp include/cudnn.h /usr/local/cuda/include
sudo cp lib64/libcudnn* /usr/local/cuda/lib64
sudo chmod a+r /usr/local/cuda/lib64/libcudnn*
```

Recompile Caffe

```
make clean
make -j4
make -j4 test
make -j4 runtest
make -j4 pycaffe
make -j4 pytest
```

Everything will take longer time compared to CPU only compilation. runtest takes about 3 times longer.