Master ST Computer Science **University of Bordeaux**
Faculty S&T
Deep Learning in Computer Vision academic year 2020 – 2021

Development Environment

Last update : 08/09/2020

During these labs, we will program mainly in Python and use the libraries: Numpy, OpenCV and Keras.
More precisely we will use the following versions of the libraries and their dependencies:
- Python 3.x [Python 2.x is not not supported anymore since 01/01/2020]
- Numpy 1.12.x or above
- OpenCV 3.2.x or above
- virtualenv
- Matplotlib 2.0.x or above
- tensorflow 2.x (and integrated Keras 2.4.0)

We will use the deep learning framework keras, integrated to tensorflow 2.x.
To have GPU acceleration for tensorflow, you need to have a Nvidia graphics card/GPU. It then requires specific version of CUDA and cuDNN. [It may be possible to have tensorflow GPU acceleration with an ATI graphics card, but this is not covered by this document]

1. CREMI

On CREMI machines all software is correctly installed in a **virtualenv** isolated python environment.
To activate this virtual environment, you have to do :
source /net/ens/DeepLearning/python3/tensorflow2/bin/activate
Your prompt should be changed and indicate that the virtual environment is active: starting by "*(tensorflow)*".
To exit the virtual environment, you can just type: **deactivate**

Most CREMI machines have a Nvidia GPU. For example, machines in room 203 have a Nvidia RTX2070 GPU. You can see the available machines and their GPUs on this page:
https://services.emi.u-bordeaux.fr/exam/?page=wol

On these machines, tensorflow with GPU acceleration is installed.
(As the installed version of CUDA is 10.1, cuDNN 7.6 have been installed to have GPU acceleration in tensorflow 2.3)

2. Personal Computer

If you want to install the development environment on your personal computer, I would recommend to have a linux distribution, Ubuntu for example. On Mac OS or Windows, it is quite easy to install a linux distribution in a virtual machine (with VirtualBox for example). It may be possible to install the development environment natively on Windows and Mac OS, but it is not covered by this document.

If you install a development environment on your personal computer, you should still check that your code is correctly working on CREMI computers. Indeed, your work will be evaluated on CREMI computers.

2.1 GPU acceleration

Both OpenCV and tensorflow can benefit from GPU acceleration for some operations.
To have GPU acceleration in tensorflow and OpenCV, you need to install CUDA and cuDNN.
To download cuDNN, you may need to register for a (free) Nvidia developer account.

In these labs, we do not use any gpu-accelerated operation from OpenCV. However, for tensorflow, if you have gpu-acceleration, deep networks training will be much faster.

Details instructions how to install CUDA and cuDNN are beyond the scope of this document.
Just know that pre-compiled binaries of tensorflow are very picky on required CUDA and cuDNN versions.
If you want to install tensorflow 2.x, you will need CUDA 10.x. For CUDA 10.1, you will need cuDNN 7.6 [7.6.4 or 7.6.5 should work]. For CUDA 10.0, you will need cuDNN 7.4.
If you have an older CUDA version, you will not be able to use a pre-compiled binary package of tensorflow.
This page https://www.tensorflow.org/install/source#linux gives tensorflow/cuDNN/CUDA tested build configurations.

2.2 Ubuntu

Ubuntu 20.04 comes with a packaged OpenCV version 4.2. Ubuntu 18.04 comes with OpenCV version 3.2. All Ubuntu version before 18.04 come with a packaged OpenCV version older than 3.2. If you want a newer OpenCV version, you will have to install OpenCV from sources (see section 2.3).

Reference instructions to install tensorflow, in particular for Ubuntu, are available here:
https://www.tensorflow.org/install/gpu?hl=fr#pip_package
In the following, we give instructions to have an installation close to the one available at CREMI.

The procedure to install all python software for the lab would be the following, where commands are in bold :

#1) install python3.7
sudo apt install python3-dev

#2) install numpy
You can install with the distribution package manager
sudo apt install python-numpy
or you could install with one python package manager (pip, anaconda, …)
sudo pip install numpy

#3) install opencv from binary packages (4.2 on Ubuntu 20.04, 3.2 on Ubuntu 18.04)
On older Ubuntu versions or for a newer OpenCV version, we would have to install it from a PPA or from sources, see section 2.3)
sudo apt install libopencv-dev
#and python opencv bindings [numpy should be installed before OpenCV python bindings]
sudo apt install python-opencv
and [optionally] contrib modules
sudo apt install libopencv-contrib-dev

#For all other python libraries, we install them in a virtual environment.
#4) install virtualenv
sudo apt install virtualenv

#4.1) create a virtualenv named (for example) tensorflow
virtualenv --system-site-packages tensorflow2

#4.2) add environment variables to the virtualenv [optional]
#For example, if CUDNN is not installed in a 'standard' path, you may have to add its path to LD_LIBRARY_PATH
echo "export LD_LIBRARY_PATH=/path/to/cudnn-7.6.5/lib64:$LD_LIBRARY_PATH" >> tensorflow2/bin/activate
#If you have installed OpenCV from sources [see section 2.3], you may need to add its python module path to the python path
echo "export PYTHONPATH=/path/to/opencv-4.4.0_install/lib/python3.7/dist-packages: $PYTHONPATH" >> tensorflow2/bin/activate

#4.3) active the virtualenv
source tensorflow2/bin/activate
#your prompt should change to begin with "(tensorflow2)"

#5) install various python packages (matplotlib, pillow, scikit-learn, gpustat, ...)
pip install gpustat
pip install pillow
pip install matplotlib
pip install scikit-learn
pip install scikit-image
pip install scikit-video

#6) install tensorflow (without GPU acceleration)
pip install tensorflow
#If you have an Nvidia GPU
- if you have installed CUDA 10.1 and cuDNN 7.6, you can install tensorflow-gpu 2.3.0
pip install tensorflow-gpu
#it will install keras [2.4.0] (which is integrated to tensorflow since tensorflow 2.0)

2.3 OpenCV

To have an up-to-date OpenCV version, (and in particular up-to-date contributed modules), I would recommend to install OpenCV from sources.

The following commands show how to install OpenCV version 4.4.0. As of 08/09/2020, 4.4.0 is the most recent available release of OpenCV.

#1) Get the latest version of opencv and opencv_contrib source code from OpenCV github repository
https://github.com/opencv/opencv/releases
https://github.com/opencv/opencv_contrib/releases
For example opencv-4.4.0.tar.gz and opencv_contrib-4.4.0.tar.gz

#2) Decompress these source code.tar.bz2
tar xvf opencv-4.4.0.tar.gz
tar xvf opencv_contrib-4.4.0.tar.gz

#3) Compile OpenCV
cd opencv-4.4.0
mkdir build
cd build

#(the following command is all on the same line)
cmake -DENABLE_FAST_MATH=ON -DBUILD_TESTS=OFF -DOPENCV_EXTRA_MODULES_PATH=../../opencv_contrib-4.4.0/modules ..
At the end of the configuration step, the list of modules that will be build is printed.
Make sure that the "python3" module will be build/
#If you have a Nvidia GPU and CUDA installed, you can enable GPU acceleration by adding -DWITH_CUDA=ON to the previous cmake command.
If CUDA is correctly detected, but not cuDNN, you can specify the actual path of cuDNN with :
-DCUDNN_LIBRARY=<path_to_libcudnn.so> -DCUDNN_INCLUDE_DIR=<path_to_cudnn_include>

make
it may take a while… If you have a CPU with several core, you should use '**make -j<number of cores>**'

sudo make install

3 Troubleshooting

- On some machines, event if the right versions of CUDA and cuDNN are installed, cuDNN initialization fails when running a tensorflow program and you get a CUDNN_STATUS_INTERNAL_ERROR error message.
(At CREMI, it happens on machines with RTX 2060 gpu and driver 440.100 ? For example, moneo on room 104, lautrec on room 204)
It seems that adding the following lines helps [on a machine with only one GPU]:
physical_devices = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)

- Tensorflow 2.x by default is very verbose. You can reduce tensorflow verbosity with the TF_CPP_MIN_LOG_LEVEL environment variable.
You can for example add the following lines before importing tensorflow/keras in your python code:
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='1' # '0' for DEBUG=all [default], '1' to filter INFO msgs, '2' to filter WARNING msgs, '3' to filter all msgs