Development Environment

Last update : 26/10/2022

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 1.19.x or above
- OpenCV 3.5.x or above
- virtualenv
- Matplotlib 3.4.x or above
- tensorflow 2.x (and integrated keras 2.x)
- pytorch 1.x

We will use the deep learning frameworks keras, integrated to tensorflow 2.x, and pytorch.
To have GPU acceleration for tensorflow/keras, you must 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]
To have GPU acceleration with Pytorch, it is better to have a Nvidia GPU. [It may work with an ATI GPU or Apple M1/M2 processor, 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 "*(tensorflow2)*".
To exit the virtual environment, you can just type: **deactivate**

Most CREMI machines have a Nvidia GPU. For example, machines in room 201 have a Nvidia GTX 3060 GPU with 12GB. You can see the available machines and their GPUs on this page:
https://services.emi.u-bordeaux.fr/exam/?page=wol

On CREMI machines, tensorflow with GPU acceleration is installed.
(As the installed version of CUDA is 11.2, cuDNN 8.1 have been installed to have GPU acceleration in tensorflow 2.5.0-2.10.0)

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. It may be possible to install the development environment natively on Windows and Mac OS, but it is not covered by this document. On Mac OS or Windows, it is quite easy to install a linux distribution in a virtual machine (with VirtualBox for example), but hardware acceleration may or may not work in such a case. Installing the environment in a virtual machine is not covered by this document.

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

2.1 GPU acceleration

OpenCV, pytorch and tensorflow/keras can benefit from GPU acceleration for some operations.
To have GPU acceleration in tensorflow and OpenCV, you need to install CUDA and cuDNN.
Pytorch python package comes with all the dependencies and is easier to install.
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 pytorch and tensorflow/keras, 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 2.5.0-2.10.0, you will need CUDA 11.2. For CUDA 11.2, you will need cuDNN 8.1. This page https://www.tensorflow.org/install/source#linux gives tensorflow/cuDNN/CUDA tested build configurations.
If you have an older CUDA version, you will not be able to use a pre-compiled binary package of tensorflow. You would have to compile tensorflow from sources. It is not covered by this document.

For the labs, you should be able to use any tensorflow 2.x version.

We suppose CUDA is already installed, in usr/local/cuda. To install cuDNN you should do the following.
First you should get cuDNN from Nvidia developer website, for example the file cudnn-11.2-linux-x64-v8.1.0.77.tgz
Then:
cd usr/local
sudo mv ~/cudnn-11.2-linux-x64-v8.1.0.77.tgz .
sudo tar xvzf cudnn-11.2-linux-x64-v8.1.0.77.tgz

2.2 Ubuntu

Ubuntu 22.04 comes with a packaged OpenCV version 4.5. Ubuntu 20.04 comes with 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 compile OpenCV from sources.
Here, we will install the last OpenCV version (4.6.0) from sources.

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.
We detail how to install tensorflow 2.9.0 (as we were not able to install 2.10.0 correctly).

Reference instructions to install pytorch are available here:
https://pytorch.org/get-started/locally/

First, 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-4.6.0.tar.gz and opencv_contrib-4.6.0.tar.gz

We suppose that you are in the directory where you want to install everything, for example /net/ens/DeepLearning/. We will install opencv in a subdirectory “install”
Then the procedure to install all python software for the lab and compile OpenCV 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 (4.6.0) from from sources
mv ~/opencv*-4.6.0.tar.gz .
tar xvzf opencv-4.6.0.tar.gz
tar xvzf opencv_contrib-4.6.0.tar.gz
cd opencv-4.6.0/
mkdir build
cd build
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
cmake .. -DCMAKE_INSTALL_PREFIX=/net/ens/DeepLearning/install/opencv-4.6.0 -DOPENCV_EXTRA_MODULES_PATH=/net/ens/DeepLearning/opencv_contrib-4.6.0/modules/ -DCMAKE_BUILD_TYPE=Release -DWITH_CUDA=ON -DBUILD_EXAMPLES=ON -DINSTALL_C_EXAMPLES=ON -DINSTALL_PYTHON_EXAMPLES=ON
-DCUDNN_LIBRARY=/usr/local/cudalib64/libcudnn.so
-DCUDNN_INCLUDE_DIR=/usr/local/cudainclude
make
make install
cd ..
#(We could remove opencv sources at this point).

#For all other python libraries, we install them in a virtual environment.
mkdir python3
cd python3

#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 [required if cuDNN is not installed in a ‘standard’ path]
echo "export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH" >> tensorflow2/bin/activate
#As we have installed OpenCV from sources, we need to add its python module path to the python path
echo "export PYTHONPATH=/net/ens/DeepLearning/install/opencv-4.6.0/lib/python3.7/site-packages:$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
#gpustat is an handy tool that replaces nvidia-smi

#6) install tensorflow 2.9.0 (without GPU acceleration)
pip install tensorflow==2.9.0
#If you have an Nvidia GPU
- if you have installed CUDA 11.2 and cuDNN 8.1, you can install tensorflow-gpu 2.5.0-2.10.0
pip install tensorflow-gpu==2.9.0
#it will install keras [2.9.0] (which is integrated to tensorflow since tensorflow 2.0)
#You should try “import tensorflow” in a python interpreter to check that the installation was successful and that you can actually use tensorflow2

#7) install pytorch
pip install torch torchvision torchaudio
This will currently install pytorch for cuda 10.2
This cuda version is not related to the cuda version installed for tensorflow. Here, cuda will be included directly in the pytorch python package. You just need to have a nvidia driver supporting cuda 10.2 [see the output of the nvidia-smi command]
You may choose to install pytorch for newer versions of cuda if your driver supports them, see https://pytorch.org/get-started/locally/

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 happened on machines with RTX 2060 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