Development environment for Deep Learning

Boris Mansencal

15/09/2023

Contents

This document details how to install a python development environment for Deep Learning (mainly tensorflow/keras & pytorch with GPU acceleration) on Linux (mainly Debian or Ubuntu).

This document only deals with Nvidia GPUs. If you have an ATI graphics card or an Apple M1/M2 processor, last versions of tensorflow and pytorch may or may not use hardware acceleration. It is not covered by this document.

To have GPU acceleration with an Nvidia GPU, you need to have a Nvidia driver, and CUDA and cuDNN libraries. Pytorch python package already contains a CUDA and cuDNN library, so there is no needd to install these libraries separately. Tensorflow/keras (up to version 2.13.x included) does not contain CUDA and cuDNN. You will have to install very specific, compatible versions to have GPU acceleration with tensorflow.

1 Installation at CREMI

As of september 2023, CREMI machines use Debian 12 stable.

Most of CREMI machines have an Nvidia GPU. See https://services.emi.u-bordeaux.fr/exam/?page=wol to list and start the machines available in each room. For example, you can see that machines in room 005 each have a GTX 1060 (with 6GB), and machines in room 202 have a RTX 3060 (with 12GB).

CREMI machines with a Nvidia GPU use the Nvidia driver 525.125.x. This driver supports up to CUDA 12.0 (according to the output of `nvidia-smi`).
The CUDA toolkit is already installed. The command `nvcc --version` indicates that it is CUDA 11.8.

We already have installed all the remaining tools (cuDNN library, tensorflow 2.13.0 and pythorch 2.0), in a python virtual environment.

To activate the virtual environment, you have to do:

```
source /net/ens/DeepLearning/python3/tensorflow2/bin/activate
```

Once activated, your prompt should change and indicate: "(tensorflow2)".

In this environment, you should be able to launch tensorflow2/keras & pytorch python code.

To deactivate the virtual environment, juste type:

```
deactivate
```

2 Installation on a personal computer

If you have a personal computer with a Nvidia GPU, you may want to install this development environment on your machine.

WARNING: even if you have a development environment on your personal computer, you should still check that your code works on CREMI machines. Indeed, your work will only be evaluated on CREMI computers.

The following documents how to install this deep learning environment on a personal computer with Linux (mainly Debian or Ubuntu).

As stated earlier, for tensorflow you will need to install specific (compatible) Nvidia driver, CUDA and cuDNN versions. All this software is proprietary and lust be installed with admin privileges (sudo). Besides, to be able to download cuDNN, you will also need to create a (free) Nvidia developer account.

We will detail how to install the same versions available at CREMI. Newer versions may be available when you will read this document.

2.1 Nvidia driver

You should first check if the Nvidia driver is not already installed. You may already have the tool `nvidia-smi` or you can for exemple check the list of installed packages `sudo dpkg --list | grep -i nvidia | grep ii`.

If the driver is not installed, you can probably install it via a package. The version you need to install will depend on your GPU. You can check what is the newer version proposed on Nvidia website https://www.nvidia.fr/Download/index.aspx?lang=fr. If you use the package *cuda* (see next section), you can skip this part (the *cuda* package will install a driver version compatible with the CUDA toolkit) Here, we detail the commands to install the version 525:

```
sudo apt update
sudo apt install nvidia-driver-525
sudo /sbin/reboot
```

Once you have rebooted, you should be able to use the command `nvidia-smi`. In the top right corner, you can see the version of CUDA supported by your driver. Here, for driver 525, CUDA 12.0. It means that you can not use any tool that requires a newer version than 12.0, but you can use tools depending on older versions.

2.2 CUDA toolkit

For a pytorch only installation, you can skip this step. Indeed CUDA toolkit is already packaged in the python package. However, you may need to know the newer CUDA version supported by your driver to choose the right pytorch package.
For tensorflow (version 2.13.x or lower), you will need to install CUDA toolkit. For tensorflow 2.14.x, it seems that they will make packages including CUDA and cuDNN available.

Here, there are two constraints on the CUDA toolkit version.

- You can not install a CUDA toolkit depending on a newer version than the one supported wy the driver (here CUDA 12.0 for driver 525).
- you need to install a CUDA toolkit version and a cuDNN version that are compatible with the tensorflow python package. You can have a look here https://www.tensorflow.org/install/source#linux, in the GPU section, to see the pairs of CUDA/cuDNN version compatible with a specific tensorflow version.

For example, you can see that tensorflow 2.12.x-2.13.x require CUDA toolkit 11.8 and cuDNN version 8.6. For tensorflow 2.5.x-2.11.x, you will need CUDA toolkit 11.2 and cuDNN 8.6.

As our driver supports CUDA 12.0, we will install the CUDA and cuDNN versions supported by the newest tensorflow version with CUDA <= 12.0, that is tensorflow 2.13.x with CUDA 11.8 and cuDNN 8.6. (On CREMI machines, CUDA toolkit 11.8 was already installed (as indcated by `nvcc --version` output).)

Instructions to install CUDA toolkit 11.8 are available here: https://developer.nvidia.com/cuda-11-8-0-download-archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=22.04&target_type=deb_local.

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-ubuntu2204.pin
sudo mv cuda-ubuntu2204.pin /etc/apt/preferences.d/cuda-repository-pin-600
wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda-repo-ubuntu2204-11-8-local_11.8.0-520.61.05-1_amd64.deb
sudo dpkg -i cuda-repo-ubuntu2204-11-8-local_11.8.0-520.61.05-1_amd64.deb
sudo cp /var/cuda-repo-ubuntu2204-11-8-local/cuda-*-keyring.gpg /usr/share/keyrings/
sudo apt-get update
```

Here we can choose:

- We install the Nvidia driver and the CUDA toolkit: `sudo apt-get -y install cuda` It will force the update of the driver you may previously have.
- Or we can install only the CUDA toolkit (considering we already have a compatible driver): `sudo apt-get install cuda-toolkit-11-8` Choose this option if you have installed the driver manually.

Once installed you should have a */usr/local/cuda*" directory (linked to the actual installed version, for example */usr/local/cuda-11.8*).

You can add */usr/local/cuda/bin* to your PATH and */usr/local/cuda/lib*" to your LD_LIBRARY_PATH. For exemple `export PATH=/usr/local/cuda/bin:$PATH` and `export LD_LIBRARY_PATH=/usr/local/cuda/lib:$LD_LIBRARY_PATH`. (You can add it to your .bashrc for exemple). You should then be able to do: `nvcc --version`.

2.3 cuDNN library

You will need to create a (free) Nvidia developer account: https://developer.nvidia.com/login. Then you can download the desired cuDNN version from https://developer.nvidia.com/rdp/cudnn-download (see "Archived cuDNN Releases" for older versions).

Here we want the version corresponding to the CUDA toolkit and the targeted tensorflow version. So cuDNN 8.6.0 for CUDA toolkit 11.8 and tensorflow 2.13.x.

So we click *Download cuDNN v8.6.0 (October 3rd, 2022), for CUDA 11.x* and download the desired package. On Debian 12 (not listed) you can get *Local Installer for Linux x86_64 (Tar)*, On Ubuntu 22.04 you can get *Local Installer for Ubuntu22.04 x86_64 (Deb)*

For Ubuntu 22.04, you can install the package with this commands:

```
sudo dpkg -i cudnn-local-repo-ubuntu2204-8.6.0.163_1.0-1_amd64.deb
```

```
sudo cp /var/cudnn-local-repo-ubuntu2204-8.6.0.163/cudnn-local-FAED14DD-keyring.gpg /usr/share/keyrings/
sudo apt-get update
sudo apt install libcudnn8-dev libcudnn8
```

For Debian, you may copy the *.tar.gz* in */usr/local* and untar it there. It will write its content in */usr/local/cuda*.

```
unxz cudnn-linux-x86_64-8.6.0.163_cuda11-archive.tar.xz
tar xvf cudnn-linux-x86_64-8.6.0.163_cuda11-archive.tar
```

2.4 Python virtual environment

Once the Nvidia driver, CUDA & cuDNN compatible versions are installed, you can now create a python virtual environment with all the deep learning python packages.

```
virtualenv --system-site-packages tensorflow2
```

It will create a directory *tensorflow2*, with a subdirectory *bin* and the script *activate* in it.

If cuDNN was not installed in a standard location (not in */usr/local/cuda*), you may need to change the *activate* script to specify its path. For example:

```
echo "export LD_LIBRARY_PATH=/net/ens/DeepLearning/stow/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/lib/:$LD_LIBRARY_PATH" >> tensorflow2/bin/activate
```

You can then activate the virtual environment:

```
source tensorflow2/bin/activate
```

(If you want to install several versions of python packages, you can create several virtual environments. But for tensorflow, currently, du to CUDA and cuDNN versions constraints, we could only install 2.12.0 and 2.13.0).

2.5 Tensorflow

In the virtual environment, you can install tensorflow package compatible with CUDA & cuDNN versions (here 2.13.0):

```
pip install tensorflow==2.13.0
```

To check the installed tensorflow version, you can do:

```
python3 -c 'import tensorflow as tf; print(tf.__version__)'
```

To check that the GPU is usable by tensorflow, you can do:

```
python3 -c 'from tensorflow.python.client import device_lib ; print(device_lib.list_local_devices())'
```

It should list all the GPUs available on your computer.

2.5.1 Troubleshooting

- On some machines, event if the right versions of CUDA and cuDNN are installed, cuDNN initialization may fail when running a tensorflow program and you get a CUDNN_STATUS_INTERNAL_ERROR error message. (At CREMI, it happened in the past on machines with RTX 2060 and driver 440.100...) It seems that adding the following lines near the beginning of the script 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
```

- If you have a computer with several GPUs, you have to take care to only use the GPUs and the GPU memory you need. By default, tensorflow or keras used with tensorflow backend uses all the memory of all the GPUs. So, if your program is not explicitly multi-GPUs, it will actually only use the first GPU but reserve the memory on all three GPUs and so the two remaining GPUs will be unavailable to other users.

There are several ways to mitigate this problem. At CUDA level, you can limit the available GPUs with the environment variable *CUDA_VISIBLE_DEVICES*. - You can set this variable before launching your program. For example:

```
 export CUDA_VISIBLE_DEVICES="2"
```

Here, only the GPU 2, that is the third GPU, is available. Or:

```
 export CUDA_VISIBLE_DEVICES="0,2"
```

Here, both GPU 0 and 2, that is the first and third GPUs, are available. - You can also set this environment variable inside your python program:

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = '2'
```

Setting *CUDA_VISIBLE_DEVICES* allows to reduce available GPUs, but tensorflow will still use all the available memory on these GPUs.

At the keras/tensorflow 2.x level, you can specify to use only the required GPU memory with the following code (to add at the beginning of your python code) :

```
import tensorflow as tf

gpus = tf.config.list_physical_devices('GPU')
if gpus:
  try:
    for gpu in gpus:
```

```
        tf.config.experimental.set_memory_growth(gpu, True)
  except RuntimeError as e:
    print(e)
```

2.6 Pytorch

pytorch comes with CUDA and cuDNN already packaged. You don't need to install specific versions. You can see the available versions here: https://pytorch.org/get-started/locally/.

In the virtual environment, for CUDA 11.8 for example, you can do:

```
pip3 install torch torchvision torchaudio --index-url https://download.
   pytorch.org/whl/cu118
```

You should then be able to list the available GPUs with the following code:

```
import torch
for i in range(torch.cuda.device_count()):
   print(torch.cuda.get_device_properties(i).name)
```

2.7 Other useful packages

In the python virtual environment, you can also install other useful python packages. For example:

```
pip install gpustat
pip install pillow
pip install matplotlib
pip install scikit-learn
pip install scikit-image
pip install scikit-video

pip install jupyter
```

...

2.8 OpenCV library [optional]

You may need to install OpenCV and a newer version than the one available with your package manager. In particular, it should be possible to compile OpenCV from scratch and have GPU acceleration for certain processing operations.

The currently last OpenCV version available is 4.8.0. You should be able to compile OpenCV with GPU acceleration on Ubuntu (22.04). On Debian 12 stable, the version of gcc is to new (12.2.0) and does not work with CUDA toolkit 11.8.

To install OpenCV 4.8.0, you need to dowload it from https://github.com/opencv/opencv/releases and you can get the additional modules from https://github.com/opencv/opencv_contrib/tags. You can the uncompress (outside the python virtual environement):

```
tar xzf opencv-4.8.0.tar.gz
tar xzf opencv_contrib-4.8.0.tar.gz
rm -f opencv-4.8.0.tar.gz opencv_contrib-4.8.0.tar.gz
```

Then if you are on Ubuntu, you can compile with:

```
cd opencv-4.8.0
mkdir build
cd build

cmake .. -DCMAKE_INSTALL_PREFIX=/net/ens/DeepLearning/stow/opencv-4.8.0 -DOPENCV_EXTRA_MODULES_PATH=/net/ens/DeepLearning/opencv_contrib-4.8.0/modules  -DCMAKE_BUILD_TYPE=Release -DWITH_CUDA=ON -DBUILD_EXAMPLES=ON -DINSTALL_C_EXAMPLES=ON -DDINSTALL_PYTHON_EXAMPLES=ON -DCUDNN_LIBRARY=/net/ens/DeepLearning/stow/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/lib/libcudnn.so -DCUDNN_INCLUDE_DIR=/net/ens/DeepLearning/stow/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/include
```

Here we specify the absolute paths to cuDNN headers and library (as there are not installed in a standard location).

You can compile with:

```
make -j 4
make install
```

(choose the numbers of threads according to you processor)

If you are on Debian 12, it will produce an error:

```
[  1%] Building NVCC (Device) object modules/core/CMakeFiles/cuda_compile_1.dir/src/cuda/cuda_compile_1_generated_gpu_mat.cu.o
In file included from /usr/include/cuda_runtime.h:83,
                 from <command-line>:
/usr/include/crt/host_config.h:132:2: error: #error -- unsupported GNU version! gcc versions later than 11 are not supported! The nvcc flag '-allow-unsupported-compiler' can be used to override this version check; however, using an unsupported host compiler may cause compilation failure or incorrect run time execution. Use at your own risk.
  132 | #error -- unsupported GNU version! gcc versions later than 11 are not supported! The nvcc flag '-allow-unsupported-compiler' can be used to override this version check; however, using an unsupported host compiler may cause compilation failure or incorrect run time execution. Use at your own risk.
      |  ^~~~~
CMake Error at cuda_compile_1_generated_gpu_mat.cu.o.Release.cmake:220 (message):
  Error generating
  /net/ens/DeepLearning/opencv-4.8.0/build/modules/core/CMakeFiles/cuda_compile_1.dir/src/cuda/./cuda_compile_1_generated_gpu_mat.cu.o
```

You can then configure to not use GPU acceleration:

```
cmake .. -DCMAKE_INSTALL_PREFIX=/net/ens/DeepLearning/stow/opencv-4.8.0 -DOPENCV_EXTRA_MODULES_PATH=/net/ens/DeepLearning/opencv_contrib-4.8.0/modules  -DCMAKE_BUILD_TYPE=Release -DWITH_CUDA=OFF -DBUILD_EXAMPLES=ON
```

```
  -DINSTALL_C_EXAMPLES=ON -DDINSTALL_PYTHON_EXAMPLES=ON -DOPENCV_DNN_CUDA=
  OFF
```

then:

```
make -j 4
make install
```

You can then add the OpenCV python module path to your python virtual environment:

```
echo "export PYTHONPATH=/net/ens/DeepLearning/stow/opencv-4.8.0/lib/python3
  .11/dist-packages:$PYTHONPATH" >> tensorflow2/bin/activate
```

You can then activate the virtual environment:

```
source tensorflow2/bin/activate
```

and should be able to display the OpenCV version:

```
python3 -c 'import cv2; print(cv2.__version__)'
```