Configuring the NVIDIA Jetson TX2's CUDA cores for OpenCV's dnn module and pyrealsense2 with the Intel Realsense Depth Camera D435

Note: If done correctly, this procedure can be completed within 4.5 hours.

Items required:

- Host computer running Ubuntu 18.04 LTS (preferred) or 16.04 LTS
- NVIDIA Jetson TX2
- Intel Realsense Depth Camera D435
- USB-A to micro USB data cable

Procedure

Setup

Flash Jetson TX2 with JetPack 4.6 on host machine with Ubuntu 18.04 LTS.
This will install the following, among other things by default:

- cuDNN 8.2.1
- CUDA 10.2
- OpenCV 4.1.1 (to be removed later since OpenCV must be compiled from source to take advantage of the CUDA cores)

Building the `librealsense` and `pyrealsense2` Libraries from Scratch

Once flash is complete, clone the `librealsense` library. This library also contains the `pyrealsense` module. To do so, run the following commands:

```
$ cd ~

$ sudo apt-get update && sudo apt-get -y upgrade
$ sudo apt-get install -y --no-install-recommends \
    python3 \
    python3-setuptools \
    python3-pip \
    python3-dev

# Install the core packages required to build librealsense libs
$ sudo apt-get install -y git libssl-dev libusb-1.0-0-dev pkg-config libgtk-3-dev

# Install Distribution-specific packages for Ubuntu 18
$ sudo apt-get install -y libglfw3-dev libgl1-mesa-dev libglu1-mesa-dev

$ git clone https://github.com/IntelRealSense/librealsense.git
$ cd ./librealsense
$ ./scripts/setup_udev_rules.sh
$ mkdir build && cd build
# Install CMake with Python bindings (that's what the -DBUILD flag is for)
# see link: https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python#building-from-source
$ cmake ../ -DBUILD_PYTHON_BINDINGS:bool=true
```

Now export `pyrealsense2` to your `PYTHONPATH` in order for 'import pyrealsense2' to work.

```
$ vim ~/.bashrc
```

Append the document with the following line:

```
export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python3.6/pyrealsense2
```

Reload the `~/.bashrc` file in your terminal session:

```
$ source ~/.bashrc
```

Test the import of `pyrealsense2` module on any python program.

```
$ python3
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyrealsense2
>>> pyrealsense2.__version__
'2.49.0'
```

Building the OpenCV library from Scratch

Now build the OpenCV library. Note that OpenCV can only take advantage of the CUDA cores from version 4.2.0 and up. So, we will switch to that branch after cloning the repository.

But first, remove the 4.1.1 version of OpenCV that is already installed.

```
$ sudo apt-get purge *libopencv*
```

Now, begin the installation.

```
$ cd ~
$ git clone https://github.com/opencv/opencv.git
$ cd opencv
$ git checkout 4.5.3
$ cd ..
$ git clone https://github.com/opencv/opencv_contrib.git
$ cd opencv_contrib
$ git checkout 4.5.3
$ cd ..
```

It is important that both `opencv` and `opencv_contrib` are in the same branch.

Now, install the relevant dependencies to configure the OpenCV's "dnn" module for NVIDIA GPU.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install build-essential cmake unzip pkg-config
$ sudo apt-get install libjpeg-dev libpng-dev libtiff-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
$ sudo apt-get install libv4l-dev libxvidcore-dev libx264-dev
$ sudo apt-get install libgtk-3-dev
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install python3-dev
```

Configure a python virtual environment for best practices.

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python3 get-pip.py
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/get-pip.py ~/.cache/pip
```

Now, you need to update your `~/.bashrc` file so that it automatically loads `virtualenv/virtualenvwrapper` when you open up the terminal.

```
$ vim ~/.bashrc
```

Insert the following:

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

Then reload the `~/.bashrc` file in your terminal session:

```
$ source ~/.bashrc
```

Now create your python virtual environment:

```
$ mkvirtualenv opencv_cuda -p python3
```

This will automatically start the virtual environment where you need to install `numpy`.

```
(opencv_cuda) $ pip install numpy
(opencv_cude) $ deactivate
```

Add the following lines to your `~/.bashrc` file:

```
export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python3.6
export OPENBLAS_CORETYPE=ARMV8
```

Then reload the `~/.bashrc` file in your terminal session:

```
$ source ~/.bashrc
```

It is paramount that you determine your NVIDIA GPU architecture version, for the TX2, it's `6.2`. This is the value that will be used for the `-D CUDA_ARCH_BIN` flag.

Now we will start building the library. First, make sure you are in the virtual environment.

```
$ workon opencv_python
```

Navigate to the `opencv` directory. And start the recipe.

```
(opencv_cuda) $ cd ~/opencv
(opencv_cuda) $ mkdir build && cd build
(opencv_cuda) $ cmake -D CMAKE_BUILD_TYPE=RELEASE \
  -D CMAKE_INSTALL_PREFIX=/usr/local \
  -D INSTALL_PYTHON_EXAMPLES=ON \
  -D INSTALL_C_EXAMPLES=OFF \
  -D OPENCV_ENABLE_NONFREE=ON \
  -D WITH_CUDA=ON \
  -D WITH_CUDNN=ON \
  -D OPENCV_DNN_CUDA=ON \
  -D ENABLE_FAST_MATH=1 \
  -D CUDA_FAST_MATH=1 \
  -D CUDA_ARCH_BIN=6.2 \
  -D WITH_CUBLAS=1 \
  -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
  -D HAVE_opencv_python3=ON \
  -D PYTHON_EXECUTABLE=~/.virtualenvs/opencv_cuda/bin/python \
  -D BUILD_EXAMPLES=ON ..

```

After running `cmake`, ensure the command executed properly by looking at the output:

```
...
--   NVIDIA CUDA:                    YES (ver 10.2, CUFFT CUBLAS FAST_MATH)
--     NVIDIA GPU arch:              62
--     NVIDIA PTX archs:
--
--   cuDNN:                          YES (ver 8.1.2)
...
```

If you get `cuDNN: NO` by any chance, a solution might be to make a change in the `opencv/cmake/FindCUDNN.cmake` file.

```
// Replace line:
file(READ "${CUDNN_INCLUDE_DIR}/cudnn.h" CUDNN_H_CONTENTS)
// With:
file(READ "${CUDNN_INCLUDE_DIR}/cudnn_version.h" CUDNN_H_CONTENTS)

```

Then start the installation.

```
(opencv_cuda) $ make -j4
(opencv_cuda) $ sudo make install
(opencv_cuda) $ sudo ldconfig
(opencv_cuda) $ deactivate
```

We are almost done, now we have to `sym-link` the OpenCV library to our python virtual environment. First, confirm the location of the OpenCV bindings. It should be in `/usr/local/lib/python3.6/site-packages/cv2/python-3.51` . You may confirm it by using the `ls` command:

```
$ ls -l /usr/local/lib/python3.6/site-packages/cv2/python-3.6
total 9996
-rw-r--r-
1 root staff 10232360 Oct 8 21:09 cv2.cpython-36m-aarch64-linux-gnu.so
```

Now that you have confirmed the location of your OpenCV bindings, you can `sym-link` them using the `ln` command as:

```
$ cd ~/.virtualenvs/opencv_cuda/lib/python3.6/site-packages/
$ ln -s /usr/local/lib/python3.6/site-packages/cv2/python-3.6/cv2.cpython-36m-aarch64-linux-gnu.so cv2.so
```

Verifying installation of OpenCV

```
$ workon opencv_cuda
(opencv_cuda) $ python3
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.5.3'
```