A guide to building OpenCV 3.4.1 from source with CUDA support

This tutorial has been put together in order to assist the students of the Autonomous Drones Lab to set up OpenCV quickly and easily. If you come across problems with these instructions, need assistance or want to suggest improvements, please address your requests to maayantamari@mail.tau.ac.il.

Environment/ Prerequisites

- Ubuntu 16.04
- CUDA 9.0
- Python 3.5/3.6
- Git
- NVIDIA GPU (Pascal/Maxwell)

Pre-installation steps

Step 1: Update pre-installed packages and install dependencies

```
sudo apt-get update

sudo apt-get install build-essential cmake pkg-config

sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev
libpng12-dev libtbb2 libtbb-dev

sudo apt-get install libavcodec-dev libavformat-dev libswscale-
dev libv4l-dev

sudo apt-get install libxvidcore-dev libx264-dev

sudo apt-get install libgtk-3-dev

sudo apt-get install libatlas-base-dev gfortran
```

- **build-essential cmake pkg-config:**
  Developer tools used to compile OpenCV 3.x

- **libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev libtbb2 libtbb-dev:**
  Libraries and packages used to read various image formats from disk

- **libavcodec-dev libavformat-dev libswscale- dev libv4l-dev libxvidcore-dev libx264-dev:**
  A few libraries used to read video formats from disk.

- **libgtk-3-dev:**
  GTK allows usage of OpenCV’s GUI features

- **libatlas-base-dev gfortran:** packages that are used to optimize various functions inside OpenCV, such as matrix operations

Installation steps

Step 1: Clone OpenCV with 3.4.1 from GitHub.

```
OPENCV_VERSION=3.4.1
git clone https://github.com/opencv/opencv.git
cd opencv
git checkout -b v${OPENCV_VERSION} ${OPENCV_VERSION}
git merge ec0bb66
git cherry-pick 549b5df
cd ..
```

Step 2: Clone OpenCV_contrib from GitHub.

```
git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib
git checkout -b v${OPENCV_VERSION} ${OPENCV_VERSION}
cd ..
```

Step 3: Set up the build.

```
cd opencv
mkdir build
cd build
```

Step 4: Run CMake command with appropriate options.

The followings should work with some adjustments.

```
 cmake -D CMAKE_BUILD_TYPE=RELEASE \
       -D CMAKE_INSTALL_PREFIX=$(python -c "import sys; print(sys.prefix)") \
       -D BUILD_EXAMPLES=ON \
       -D BUILD_opencv_python2=OFF \
       -D BUILD_opencv_python3=ON \
       -D PYTHON_DEFAULT_EXECUTABLE=/usr/bin/python3.5 \
       -D INSTALL_C_EXAMPLES=OFF \
       -D INSTALL_PYTHON_EXAMPLES=ON \
       -D ENABLE_FAST_MATH=ON \
       -D CUDA_FAST_MATH=ON \
       -D CUDA_GENERATION=Pascal \
       -D WITH_FFMPEG=1 \
       -D WITH_CUDA=ON \
       -D WITH_CUBLAS=1 \
       -D WITH_OPENGL=ON \
       -D WITH_TBB=ON \
       -D WITH_LAPACK=OFF \
       -D OPENCV_EXTRA_MODULES_PATH="../../opencv_contrib/modules" \
       -D PYTHON3_EXECUTABLE=$(which python) \
       -D PYTHON3_INCLUDE_DIR=$(python -c "from distutils.sysconfig import
get_python_inc; print(get_python_inc())") \
       -D PYTHON3_LIBRARY=$(python -c "from distutils.sysconfig import
get_python_lib; print(get_python_lib())") \
       -D PYTHON3_PACKAGES_PATH=$(python -c "from distutils.sysconfig import
get_python_lib; print(get_python_lib())") \
       -D PYTHON3_NUMPY_INCLUDE_DIRS=$(python -c "from distutils.sysconfig
import get_python_lib; print(get_python_lib())")
```

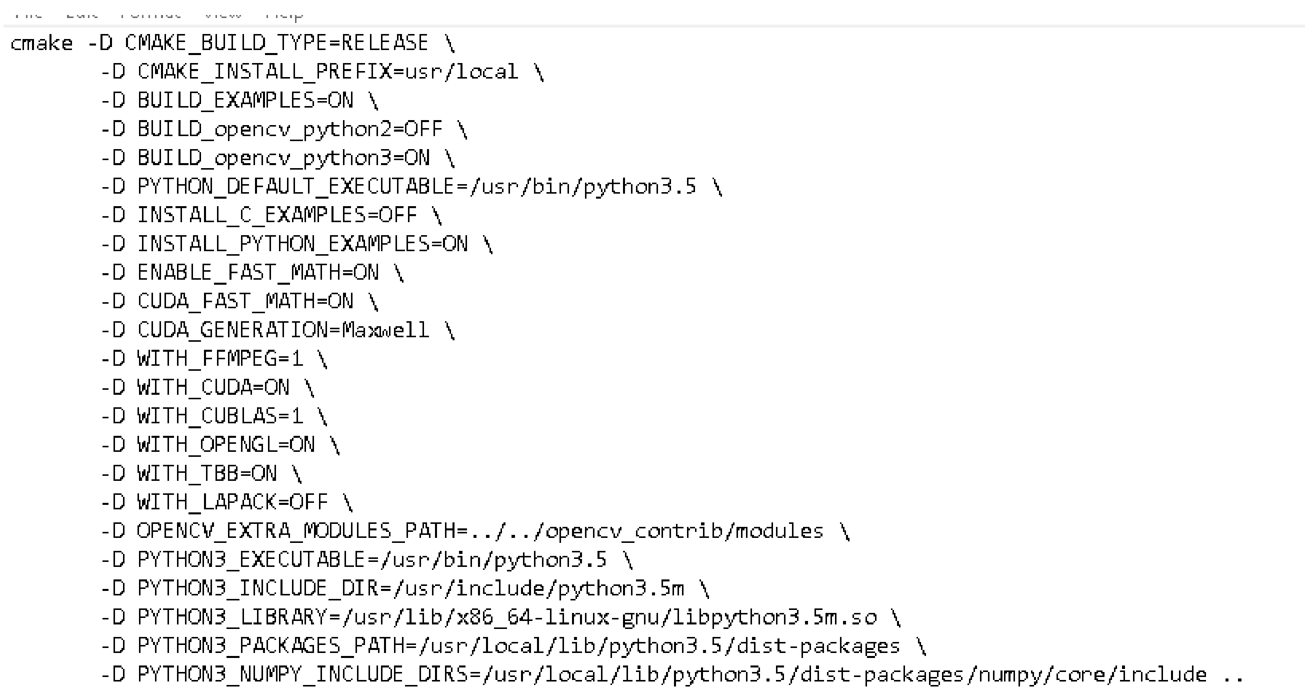
Adjust the followings:

- Replace CUDA_GENERATION with the proper one for your GPU. CUDA_GENERAT|ION is used for specification of Auto, Fermi, Pascal, Maxwell, Volta etc. It limits the code generation only for specific architecture.

- Check all paths defined here and see that they lead to the correct place. Meaning- all lines with **=$(something... python) or =$(python something...)**
  Note: It is highly recommended to replace these commands with explicit paths, especially if you have more than one version of python installed.

- check the path to **PYTHON_DEFAULT_EXECUTABLE=path/to/executable**

Where should these paths lead?
In the next image, you can see my CMake file with the explicit paths that resulted in a successful build.



```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
       -D CMAKE_INSTALL_PREFIX=usr/local \
       -D BUILD_EXAMPLES=ON \
       -D BUILD_opencv_python2=OFF \
       -D BUILD_opencv_python3=ON \
       -D PYTHON_DEFAULT_EXECUTABLE=/usr/bin/python3.5 \
       -D INSTALL_C_EXAMPLES=OFF \
       -D INSTALL_PYTHON_EXAMPLES=ON \
       -D ENABLE_FAST_MATH=ON \
       -D CUDA_FAST_MATH=ON \
       -D CUDA_GENERATION=Maxwell \
       -D WITH_FFMPEG=1 \
       -D WITH_CUDA=ON \
       -D WITH_CUBLAS=1 \
       -D WITH_OPENGL=ON \
       -D WITH_TBB=ON \
       -D WITH_LAPACK=OFF \
       -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
       -D PYTHON3_EXECUTABLE=/usr/bin/python3.5 \
       -D PYTHON3_INCLUDE_DIR=/usr/include/python3.5m \
       -D PYTHON3_LIBRARY=/usr/lib/x86_64-linux-gnu/libpython3.5m.so \
       -D PYTHON3_PACKAGES_PATH=/usr/local/lib/python3.5/dist-packages \
       -D PYTHON3_NUMPY_INCLUDE_DIRS=/usr/local/lib/python3.5/dist-packages/numpy/core/include ..
```

You can also open python and **import sys**, then **print(sys.path)** to get an idea of where you should look in your system.

```
Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0] on linux
```

```
Type "help", "copyright", "credits" or "license" for more
information.
>>> import sys
>>> print(sys.path)
```

Note: python<version> files vs python<version>m files
As you can see, some of the paths refer to python3.5 and some to python3.5m. Which should you choose if you have both? They should be hard links of each other. It should not matter which one of them you use.

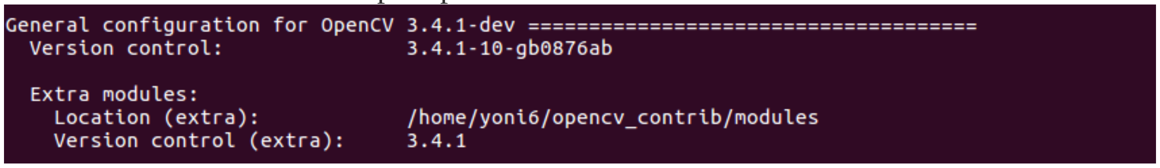
Step 5: Make the build.

```
make -j $(nproc)
```

Before moving on to the next step you should verify in the printout at the end of the build process that NVIDIA CUDA is enabled and that all the paths are correct.
If they are not correct, you should delete the build folder and do step 5 again after fixing the CMake commands you are using.

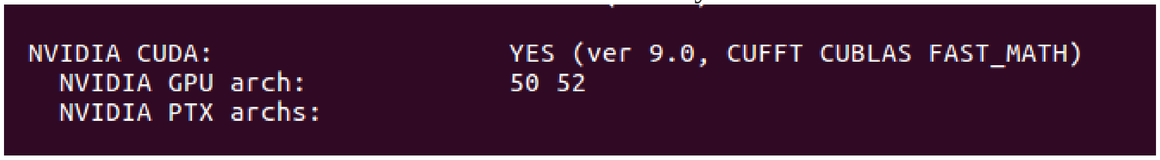
What are you looking for in the print?

You want to see the correct version of OpenCV. If you cloned OpenCV-contrib you want to see that the extra modules path point to the correct location.



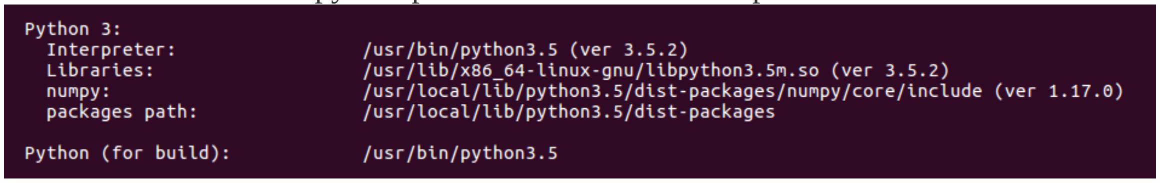
```
General configuration for OpenCV 3.4.1-dev =====================================
  Version control:               3.4.1-10-gb0876ab

  Extra modules:
    Location (extra):            /home/yoni6/opencv_contrib/modules
    Version control (extra):     3.4.1
```

You want to see that NVIDIA CUDA is marked with yes and the correct CUDA version



```
  NVIDIA CUDA:                   YES (ver 9.0, CUFFT CUBLAS FAST_MATH)
    NVIDIA GPU arch:             50 52
    NVIDIA PTX archs:
```

You want to see that all python paths refer to their correct place and version.



```
  Python 3:
    Interpreter:                 /usr/bin/python3.5 (ver 3.5.2)
    Libraries:                   /usr/lib/x86_64-linux-gnu/libpython3.5m.so (ver 3.5.2)
    numpy:                       /usr/local/lib/python3.5/dist-packages/numpy/core/include (ver 1.17.0)
    packages path:               /usr/local/lib/python3.5/dist-packages

  Python (for build):            /usr/bin/python3.5
```

Step 6: Install OpenCV

If the build was successful, you can proceed to installing OpenCV.

```
sudo make install
```

Step 7: Verify OpenCV

You should verify that the installation was successful from Python. Open python, **import cv2** and then print **cv2**.**getBuildInformation**().

```
Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import cv2
>>> print(cv2.getBuildInformation())
```

If everything is ok you should see the same paths and NVIDIA details as in step 6.
If you get an import error or the build information is not correct, you should uninstall and try again.

```
sudo make uninstall
```

Note: Wrong paths in getBuildInformation even though they seemed fine before
It might be that while in step 6 you saw that all paths are correct and NVIDIA CUDA was enabled, in this step when you check from within python, you will see different paths then the ones you set up. This is likely to be the result of many attempts to install OpenCV, if you had errors in your previous attempts or maybe you have a version previously installed that is interfering.
Try first to uninstall as explained above. Then open python and check if when you import OpenCV you get an import error. If you can import OpenCV after you uninstalled then you should try uninstalling with apt-get (assuming you have a version that was installed with apt-get) and maybe consider manually deleting relevant files until you get an import error when you try to import OpenCV.
However, if you are considering manually deleting OpenCV, read about it first, manually deleting may result in other issues and problems with python/Ubuntu installations.

Step 8: Add environment variables to .bashrc

Go to:

```
cd $(python -c "import sys; print(sys.prefix)") && pwd
```

Remember that this is the same command used in the CMake file line 2-**CMAKE_INSTALL_PREFIX**. So if you needed to change it for step 5, use the path you used with CMake.
Now, you want to find the path to a file called
cv2.cpython-< python_version>-x86_64-linux-gnu.so

```
find . -iname cv2.cpython-36m-x86_64-linux-gnu.so
./lib/python3.6/site-packages/cv2.cpython-36m-x86_64-linux-gnu.so
```

Now that you have the full path to the file, you want to export it in .bashrc
For example, open with nano

```
nano ~/.bashrc
```

Add to the end of the file:

```
export
PYTHONPATH=$PYTHONPATH:/path/to/lib/your_version_of_python/site-
packages
export LD_LIBRARY_PATH="/path/to/lib/:$LD_LIBRARY_PATH"
```

Save your changes and exit. Do not forget to close the terminal or source the changes you made.

```
source ~/.bashrc
```

Sources:

This guide is based mostly on this guide:

http://leadtosilverlining.blogspot.com/2018/09/build-opencv-341-with-cuda-90-support.html