Installing ML tools for Ubuntu 16.04 Linux on Dell Precision 5820 Tower PC

(ver 1.8, 11 Jan 2019)

Daniele Bagni

DSP / ML Specialist for EMEA

Contents

XILINX

HISTORY

- **1v4.** Added the WARNING page and the new Sections 10.0 and 11.0
- **1v6.** Corrected some typo errors, added PuTTY, added NCCL1.0 (thanks to Giovanni Guasti and Mark Harvey for their review).
- **1v7.** Updated section 11 to the newest DeePhi dnndk_v2.07 release (thanks to Giovanni Guasti and Antonello Di Fresco and Jon Cory for their review).
- **1v8.** Updated to the newest DeePhi 2.0.8 DNNDK release. Added Section 12 about AWS. Almost rewritten Section 9 (and added Subsection 9.3 to solve two very recent errors that never appeared in the past), 10 and 11.

WARNING: read it before starting, please!

Installing **Caffe framework for ML development** can be really a painful and time consuming process, especially if you want to use your NVIDIA GPU to fasten the simulation time, which is basically mandatory as Caffe compiled with CPU is extremely slow and you cannot practically use it with "normal" CNNs. This is why I have written this document, after having installed Ubuntu 16.04 from scratch at least 5-6 times and Caffe at least 10-12 times: I would like you to save your time and not feel the pain I had to. Therefore it is very important that you follow what written here in a disciplined way.

Note that I have selected **Python2.7 and OpenCV 2.x** to make life easier, you can change them at your own risk.

Sections 2 to 5 of this document explain you how to install correctly Ubuntu 16.04, you cannot skip them.

Sections 6 - 7 explain you how to install your GPU driver and the CUDA and cuDNN libraries, you cannot skip these sections too. In particular in this document I have selected **CUDA 8.0** libraries as they are compatible with Caffe, TensorFlow and DeePhi tools. But nobody prevents you to install both CUDA 8.0 and 9.0, all in all they are placed in different folders and can live together (I did it in another PC). Most of the problems you might find about these libraries, once installed, is to make them visible, so check the environmental variable **`$LD_LIBRARY_PATH`** in your **.bashrc** Linux configuration file.

Sections 8 - 9 explain you how to install Python virtual environment and Caffe (or its forks). These are the most difficult sections. Please skip them the first time and go directly to Section 10.

Section 10 is a shortcut based on a script that should execute everything for you: that script is just doing automatically what explained in the Sections 8 and 9. If it fails, you have to go to those Sections and see in more details what to do step by step. Note that Section 10 relies on a small archive into which I have put the correct **Makefile** and **Makefile.config** files to avoid you any further editing on original Caffe files.

Section 11 explains you how to install **DeePhi tool for quantization**. You cannot skip it.

Section 12 explains how **to setup the AWS** and how copying there the DeePhi Host tools. You cannot skip it.

In case of errors, I strongly recommend you to copy and paste the error message on Google: 90% of the time a page from Stack Overflow website (https://stackoverflow.com) will appear with possible solutions. Also check the other fundamental environmental variable: **`$PYTHONPATH`**.

Last but not least, please note that there is a big difference in installing a Linux package with "**`sudo apt-get install`**" or with "**`pip2 install`**": the first command puts everything in the `/usr/lib` and `/usr/local/lib/` which are folders visible by any user, but the second puts everything in a python virtual environment which is hidden and it becomes visible only if you activate it with the `workon` command. So packages installed with the first command must be installed only once forever, but packages installed with the second command must be installed per each python virtual environment you have.

I tried to do at my best to manage different python virtual environments in a clean way, I hope you will enjoy it.

Daniele Bagni

1.0 Machine Learning SW specification

This document tries to explain how installing ML SW environments as **Caffe** and **TensorFlow/Keras** on a **Ubuntu 16.04 LTS** (Xenial) Linux PC, in particular in this document I will use a Dell Precision 5820 Tower Desktop PC, but I have successfully adopted the same procedure also for Dell LapTops. Instead of Caffe you will install **Ristretto**, which is a Caffe public domain fork on GitHub (with the same instructions, Xilinx employee could even install Caffe-SSD-Ristretto, which is an internal Xilinx GitHub project suitable for Object Detection with SSD and not available out of Xilinx).

The dependencies of Caffe on a lot of Linux packages can make the install and compilation processes very painful and time consuming.

All the material of this document is based on the following main webpages (here listed for your reference) which are the best explained ones among the several pages available on Internet on this subject, according to my experience.

- https://gl-research.com/caffe/ristretto/wikis/home
- https://github.com/BVLC/caffe/wiki/Ubuntu-16.04-or-15.10-Installation-Guide
- https://www.pyimagesearch.com/2016/10/24/ubuntu-16-04-how-to-install-opencv/
- https://www.pyimagesearch.com/2017/09/27/setting-up-ubuntu-16-04-cuda-gpu-for-deep-learning-with-python/
- http://docs.python-guide.org/en/latest/dev/virtualenvs/
- https://github.com/BVLC/caffe/wiki/OpenCV-3.2-Installation-Guide-on-Ubuntu-16.04
- https://gist.github.com/arundasan91/b432cb011d1c45b65222d0fac5f9232c
- https://groups.google.com/forum/#!topic/ristretto-users
- https://github.com/BVLC/caffe/issues/5793

Furthermore, DeePhi requires the host (PC) side of its ML toolchain to be compatible with:

- ubuntu 16.04 + CUDA 8.0 + cuDNN v7.0.5
- ubuntu 16.04 + CUDA 9.0 + cuDNN v7.0.5
- ubuntu 16.04 + CUDA 9.1 + cuDNN v7.0.5

Caffe 1.0 BVLC, Caffe-Ristretto and Caffe-SSD-Ristretto require:

- Python 2.7

TensorFlow-GPU 1.4.1 requires:

- either Python 2.7 or Python 3.5
- CUDA 8.0

In my case I have selected CUDA 8.0 because I could have both TensorFlow and Caffe in the same Python2.7 virtual environment.

2.0 Dell Precision 5829 Tower PC specifications

Precision 5820 Tower XCTO Base

Components

1	Intel Xeon W-2175 2.5GHz, 4.3GHz Turbo, 14C, 19.25M Cache, HT, (140W) DDR4-2666
1	No Additional Cable Requested
1	Precision 5820 Tower 950W Chassis
1	64GB (4x16GB) 2666MHz DDR4 RDIMM ECC
4	No Hard Drive
2	3.5" 4TB 5400rpm SATA Hard Drive
1	Integrated Intel AHCI SATA chipset controller (8x 6.0Gb/s), SW RAID 0,1,5,10
1	CPU Heatsink 5820 Tower
1	No Optical
1	Slim filler panel (no opt.)
1	UK/Irish/MY/SG/HK/Bangladesh/Pakistan/Sri Lanka/Brunei Power Cord
1	NVIDIA Quadro P6000, 24GB, 4 DP, DL-DVI-D (5820T)
1	No Additional Network Card Selected (Integrated NIC included)
1	No Stand included
1	Dell Wired Mouse MS116 Black
1	Dell Multimedia Keyboard - UK (QWERTY) - Black
2	Dell Precision Optimizer (DPO)
1	SupportAssist
1	SATA/SAS Hard Drive/Solid State Drive
1	No Raid
1	Ship Material Tower 5820,7820
1	Shipping Material for TPM

Software

1	Boot drive or storage volume is greater than 2TB (select when 3TB/4TB HDD is ordered)
1	No Cyberlink Media Suite
1	Precision T5820/T7820/T7920 Resource DVD
1	Windows 10 Pro for Workstations (4 Cores Plus) English
1	Windows 10 Pro OS Recovery 64bit - DVD
1	Microsoft Office 30 Day Trial - Excludes Office License
1	Intel vPro Technology Enabled
1	Dell Applications for Windows
1	Dell Developed Recovery Environment
1	McAfee Security Center 30 day trial, Digital Delivery

Service

1	Base Warranty
1	3Y Basic Onsite Service - Minimum Warranty
1	3Yr ProSupport Plus and Next Business Day On-Site Service
1	3Yr ProSupport Plus Keep Your Hard Drive
1	3Yr ProSupport Plus Accidental Damage Protection Commencing on Invoice Date

3.0 BIOS setup

Enter into the BIOS menu of the PC by pushing **F2 or F12 key** during the PC bootstrap and set the items illustrated in the following Figures 1, 2, 3, 4, 5:

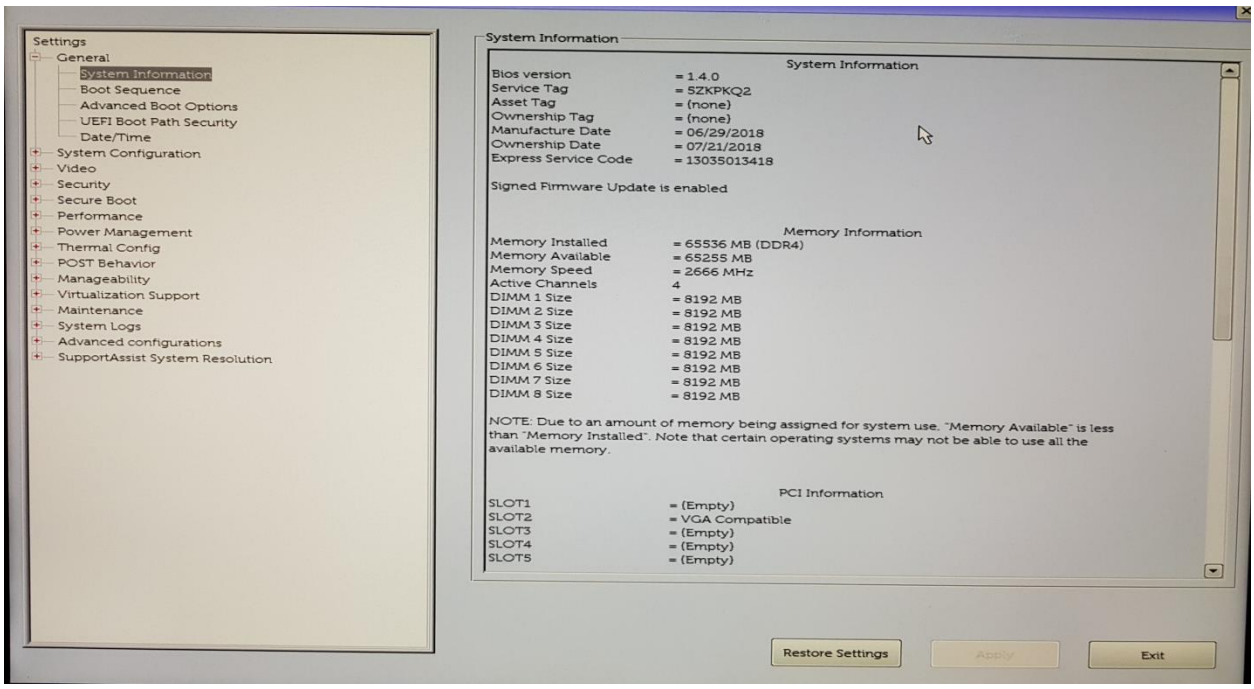

Figure 1: BIOS setup, System Information page 1

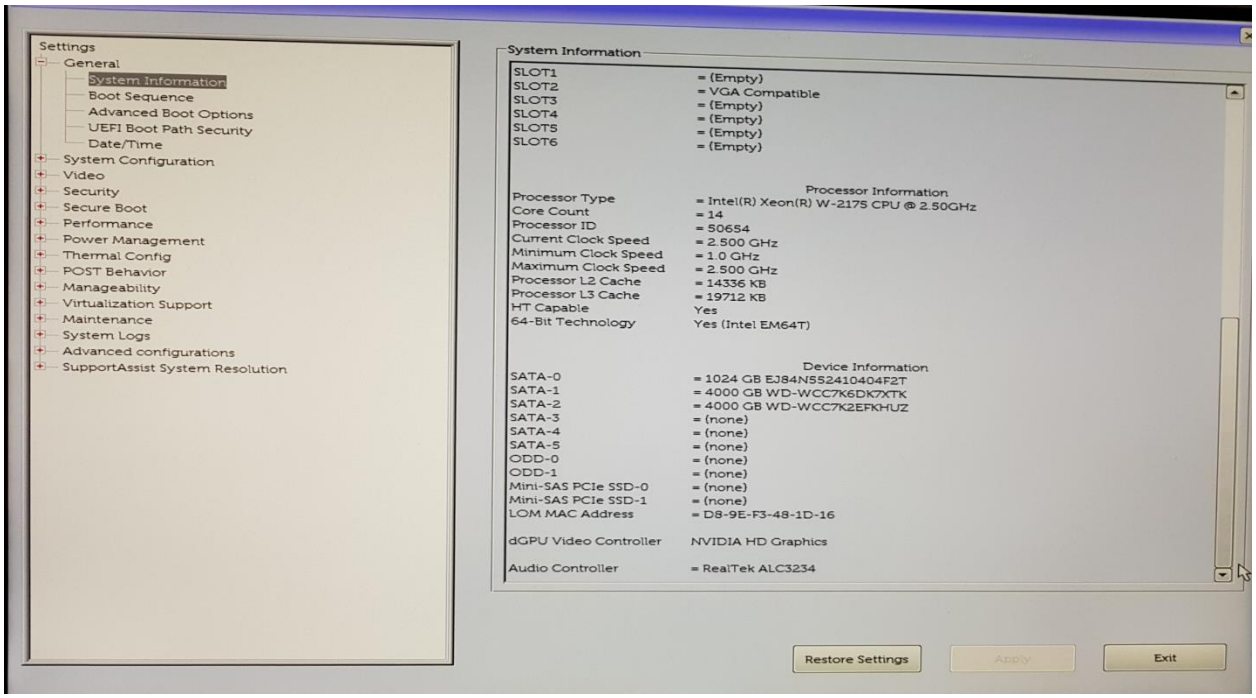

Figure 2: BIOS setup, System Information page 2

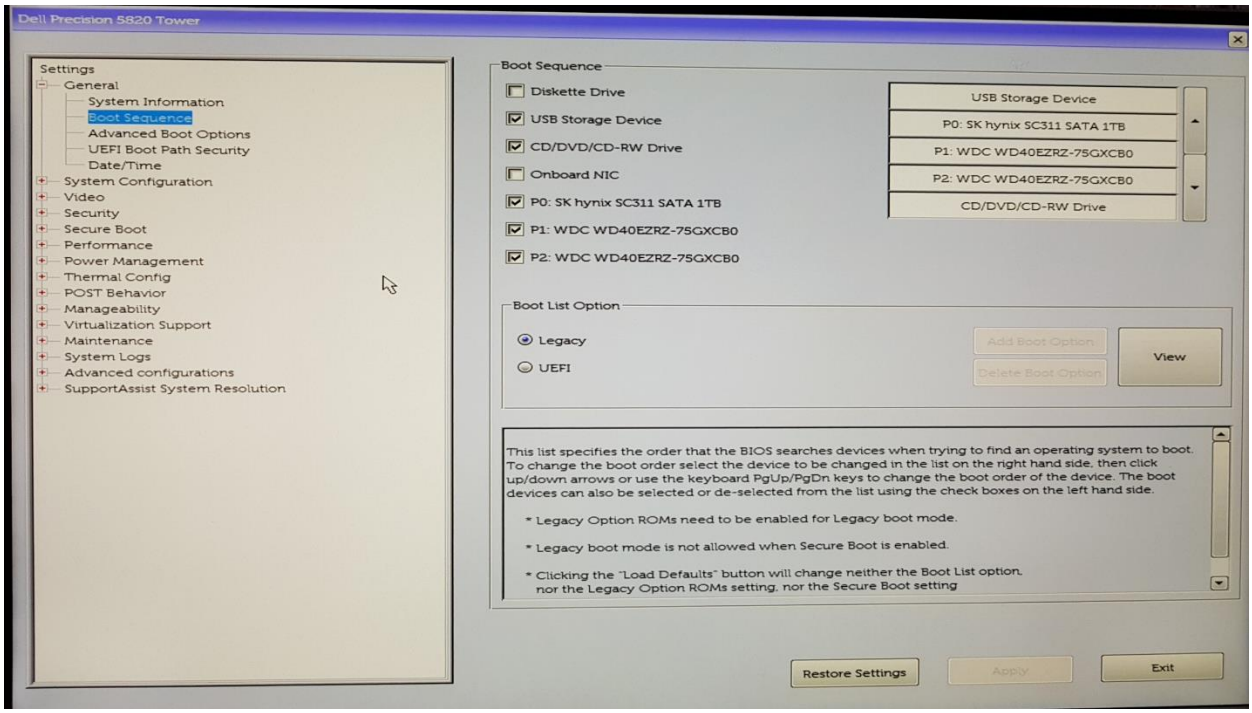

Figure 3: BIOS setup, Boot Sequence page

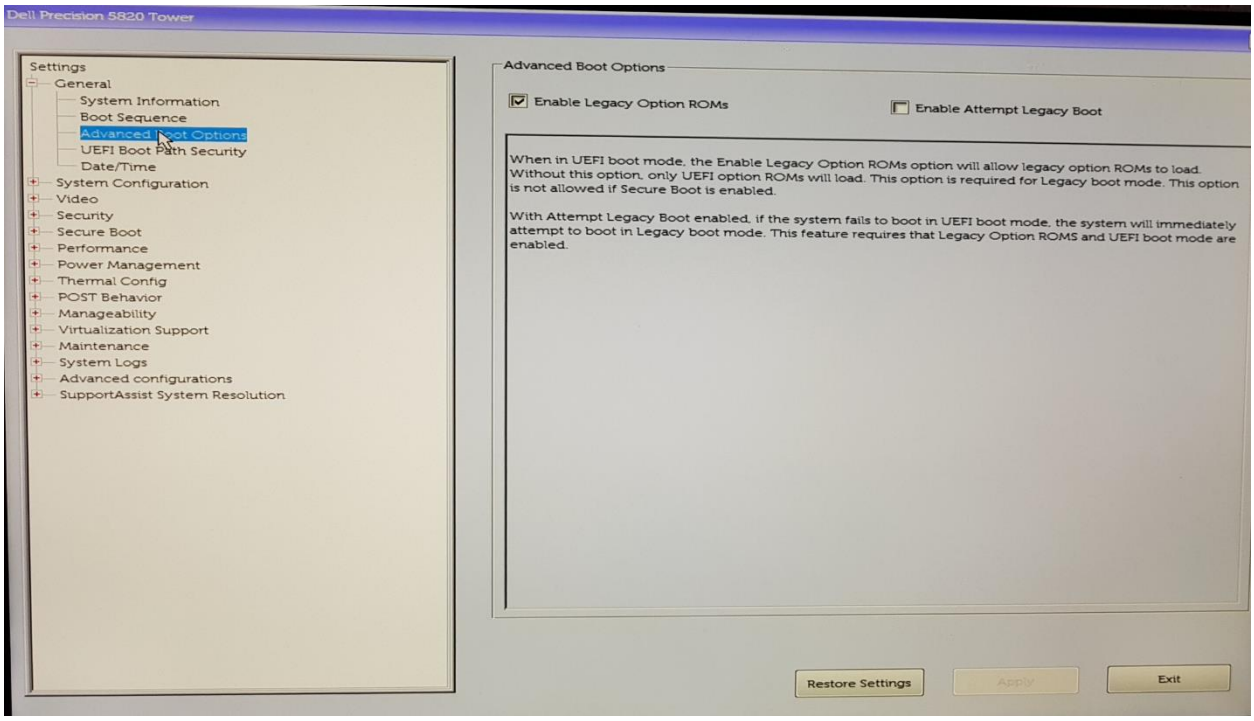

Figure 4: BIOS setup, Advanced Boot Options page

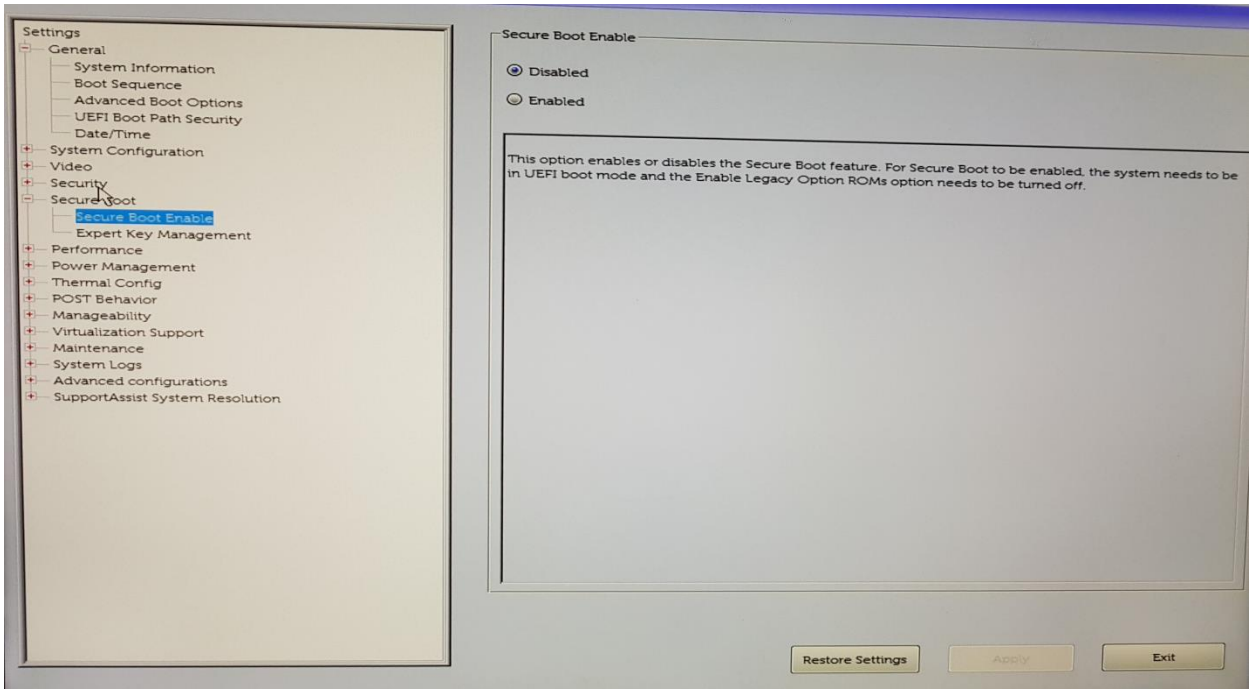

Figure 5: BIOS setup, Secure Boot Enable page

4.0 Booting the PC from a Linux USB stick

1) Download the **ubuntu-16.04.1-desktop-amd64.iso** image from http://old-releases.ubuntu.com/releases/16.04.1/
2) Prepare a USB stick to boot the PC from it with Ubuntu, according to what written here https://help.ubuntu.com/community/Installation/FromUSBStick.
   You first have to download the Linux Live USB creator executable from http://www.linuxliveusb.com/en/help/guide/preparation and set it according to what illustrated in Figure 6.

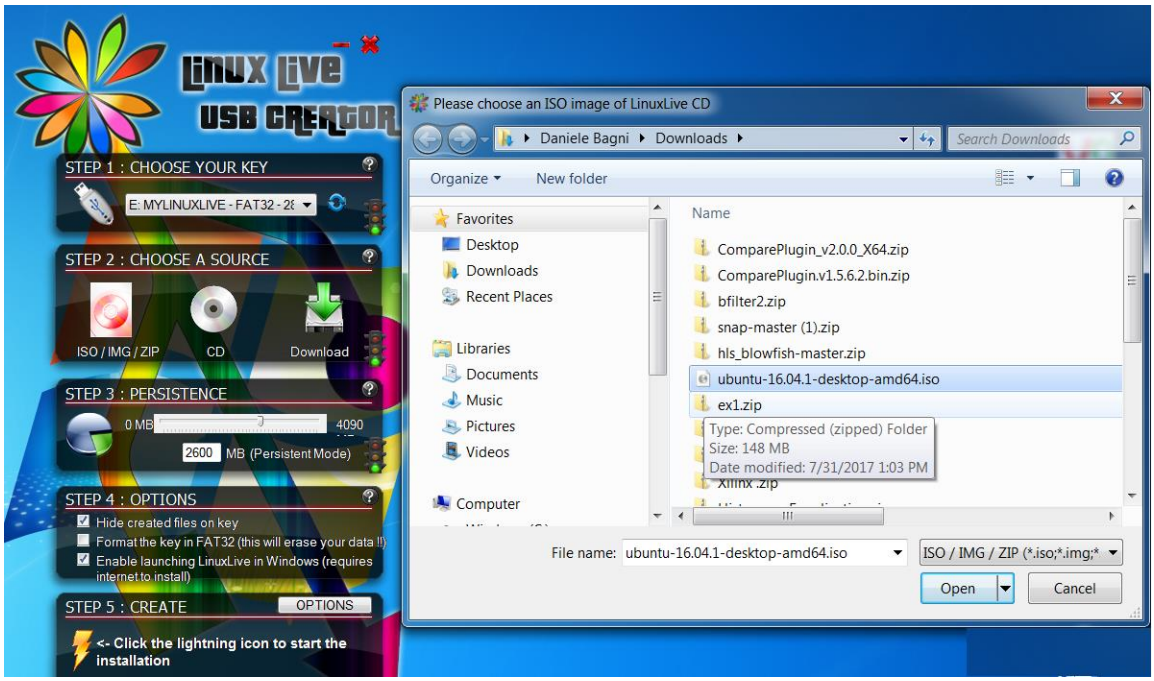

Figure 6: settings of Linux Live USB Creator GUI.

5.0 First problems: VGA resolution and no Ethernet connection

I immediately had 2 major problems after having installed Ubuntu overwriting completely Windows-10: no connectivity with my Ethernet WLAN board (so basically no connection to Internet) and the default monitor forced to VGA resolution which makes impossible to use the GUI, being the desktop image size much larger than the whole screen of my monitor (so you cannot click on some tags of the various windows). I had to switch to terminal command line with **CTRL-ALT-F1** keys.

Before solving the Ethernet connection missing, I did the following actions, which are anyway good in general.

The Ethernet port name for the machine is using a new format instead of the traditional Linux names “eth0, eth1, eth2”and so on.

I therefore looked at https://www.xilinx.com/support/answers/60510.html and https://askubuntu.com/questions/767786/changing-network-interfaces-name-ubuntu-16-04

and I ended up with the following solution:

- the file `/etc/udev/rules.d/70-persistent-net.rules` now has to be created manually. The line for fixing the interface name of the NIC with for example a MAC address like "02:01:02:03:04:05" to "eth0" is now:

`SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="02:01:02:03:04:05", ATTR{dev_id}=="0x0", ATTR{type}=="1", NAME="eth0".`
This line looks nearly the same as in Ubuntu 14.04 LTS with one slight difference: In Ubuntu 14.04, there was the additional condition `KERNEL=="eth*"`. For some reason, this does not work in Ubuntu 16.04 LTS. If this additional condition is present, the whole line is ignored and you are back to the default behavior (as specified in `80-net-setup-link.rules`).

- Modify the file **`/etc/network/interfaces`** as shown in Figure 7 screenshot (right side).

- edit your **`/etc/default/grub`** changing the line from
  - `GRUB_CMDLINE_LINUX=""` to
  - `GRUB_CMDLINE_LINUX="net.ifnames=0 biosdevname=0"`
  - and, finally:
  - `sudo update-grub`
  - and reboot your system:
  - `sudo reboot`

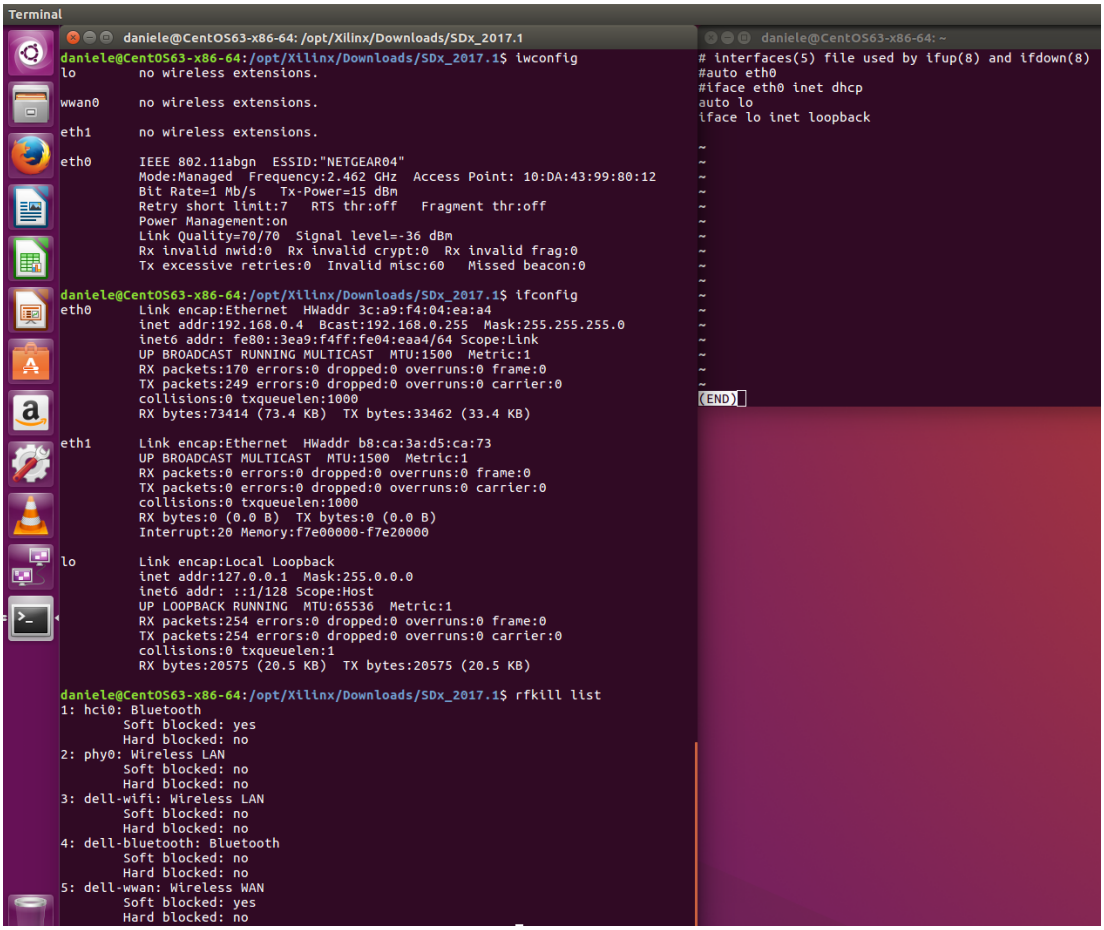

Figure 7: on the left some interesting Linux "radio" commands, on the right the /etc/network/interfaces file content

5.1 Ethernet Connection

The following Linux commands are useful to identify networks connectivity problems

```
ifconfig
dmesg | grep -e eth -e e1000
lspci | grep Ethernet
lspci -nnk
sudo lshw –C network
```

I got them by looking at the following websites:

- https://askubuntu.com/questions/650953/intel-e1000e-ethernet-not-working
- https://ubuntuforums.org/showthread.php?t=2332332
- https://ubuntuforums.org/showthread.php?t=1314693

In particular, the last command returned **-network UNCLAIMED description: Ethernet controller* from which it is evident that the installed Ubuntu missed the network interface driver for my **Intel Ethernet Connection I219-LM** adapter. I therefore browsed on

- https://downloadcenter.intel.com/product/82185/Intel-Ethernet-Connection-I219-LM
- https://downloadcenter.intel.com/download/22283/Ethernet-Intel-Ethernet-Adapter-Complete-Driver-Pack?product=82185
- https://downloadcenter.intel.com/download/27840/Ethernet-Intel-Ethernet-Adapter-Complete-Driver-Pack

and I downloaded the Driver Pack OS independent release 23.1 (463 MB size), as shown in Figure 8.

I copied such zip file in the Ubuntu PC, I unzipped it and entered into the PRO1000\Linux folder as shown in Figure 9. To be honest, before doing it I tried other folders, but instructions were not clear and I decided to look at this assuming “1000” had a relationship with “e1000e”

Then I entered into the **e1000e-3.4.0.2.tar** archive, in the subdirectory **src** and I launched the command

```
make install
sudo modprob e1000e
```

and magically the Internet connection started to work correctly.

After that I run the classic commands (which took quite a while) to complete the install of Ubuntu 16.04

```
sudo apt-get update
sudo apt-get upgrade
```

and the NIC (Network Interface Controller) started to work fine giving me wired access to Internet.

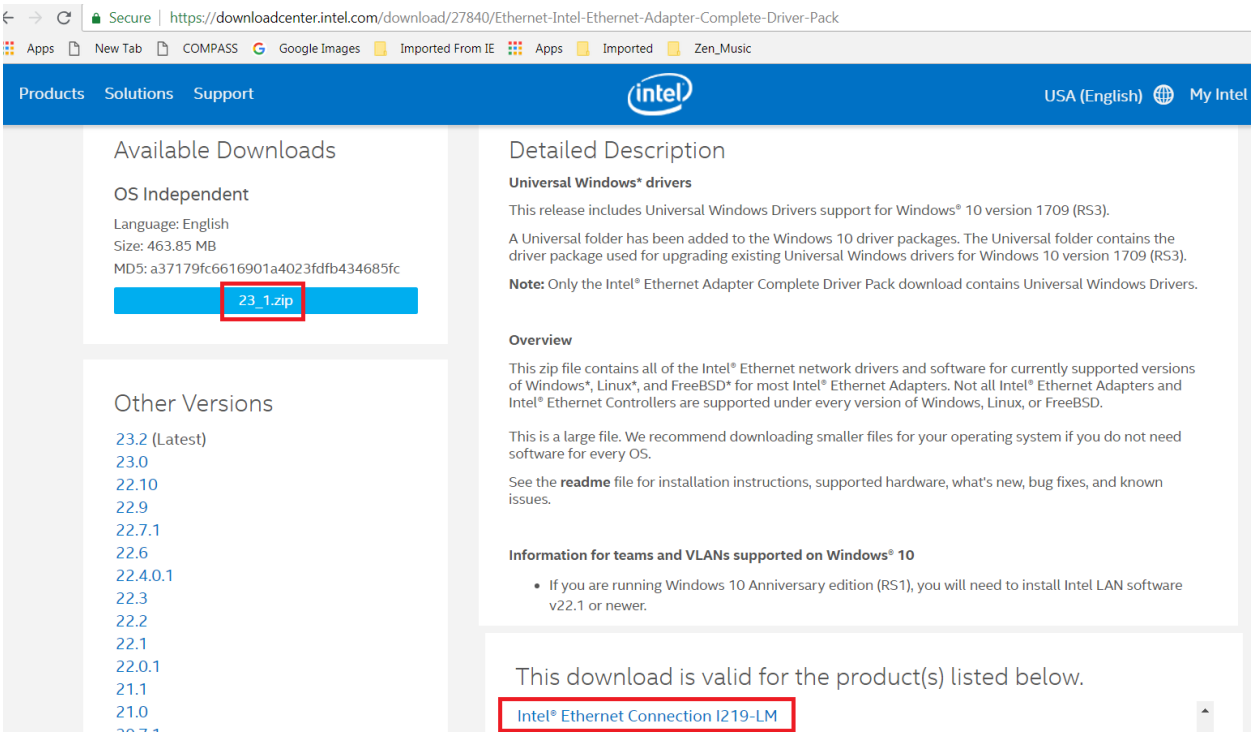

Figure 8: Intel Ethernet Adapter driver

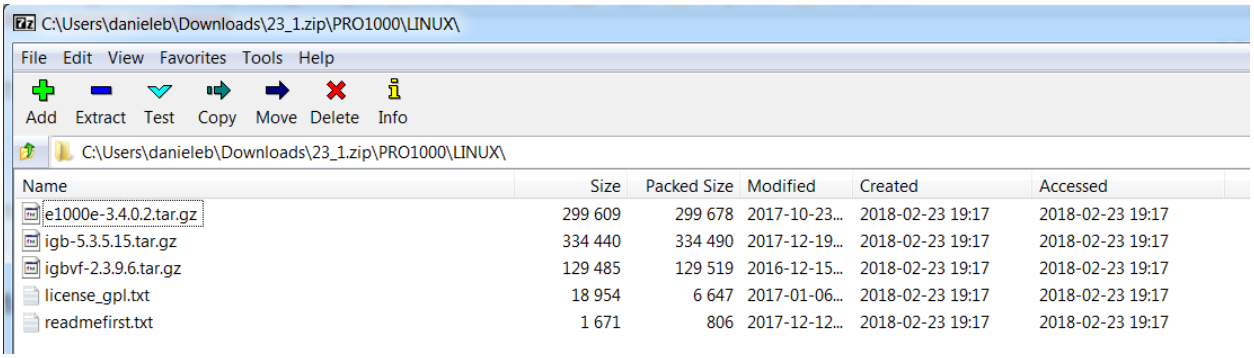

Figure 9: E1000 virtual network adapter for Intel 82545EM Gigabit Ethernet NIC

5.2 VGA default monitor

Looking at the below website https://askubuntu.com/questions/901326/how-to-start-a-fail-safe-graphics-session-with-mouse-web-browser

I could bring up the GRUB menu after the BIOS had finished its loading by pressing the **SHIFT key**, and I selected the **advanced options** (see Figure 10)**, recovery mode** (see Figure 11) and from there the **Resume normal boot** (see Figure 12, after that I got an 800x600 SVGA resolution that enabled me to work decently with the Ubuntu desktop GUI.

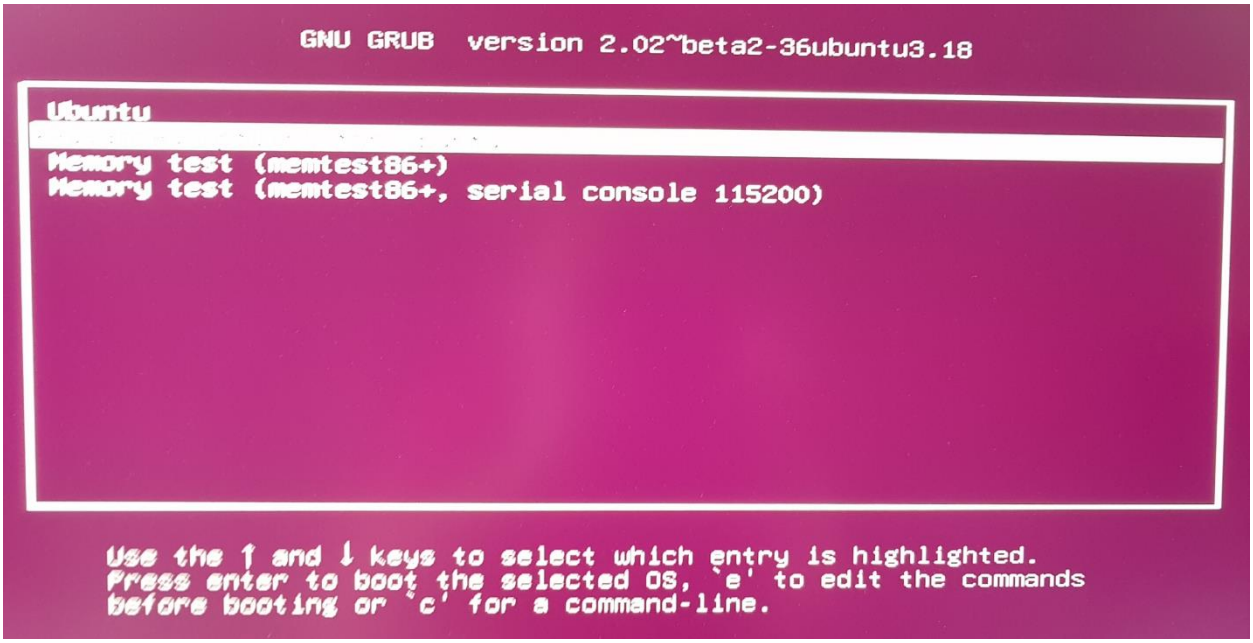

Figure 10: GRUB Ubuntu advanced options

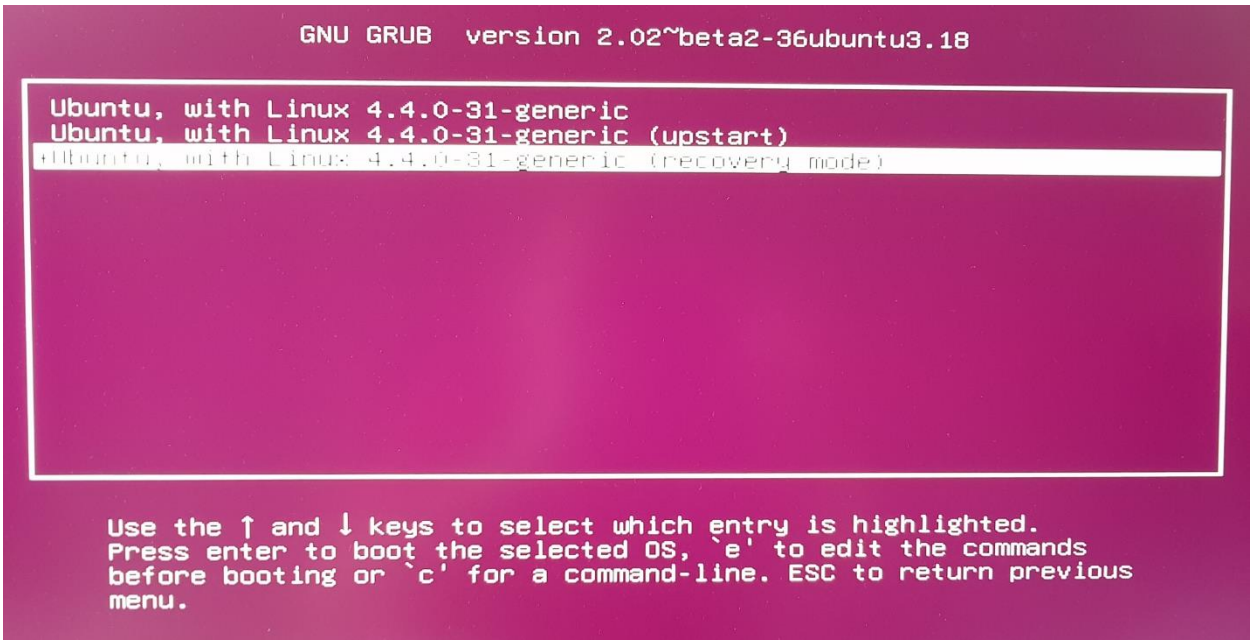

Figure 11: GRUB recovery mode

After having looked at the following pages:

- https://askubuntu.com/questions/564049/desktop-bigger-than-screen
- http://ubuntuhandbook.org/index.php/2017/04/custom-screen-resolution-ubuntu-desktop/

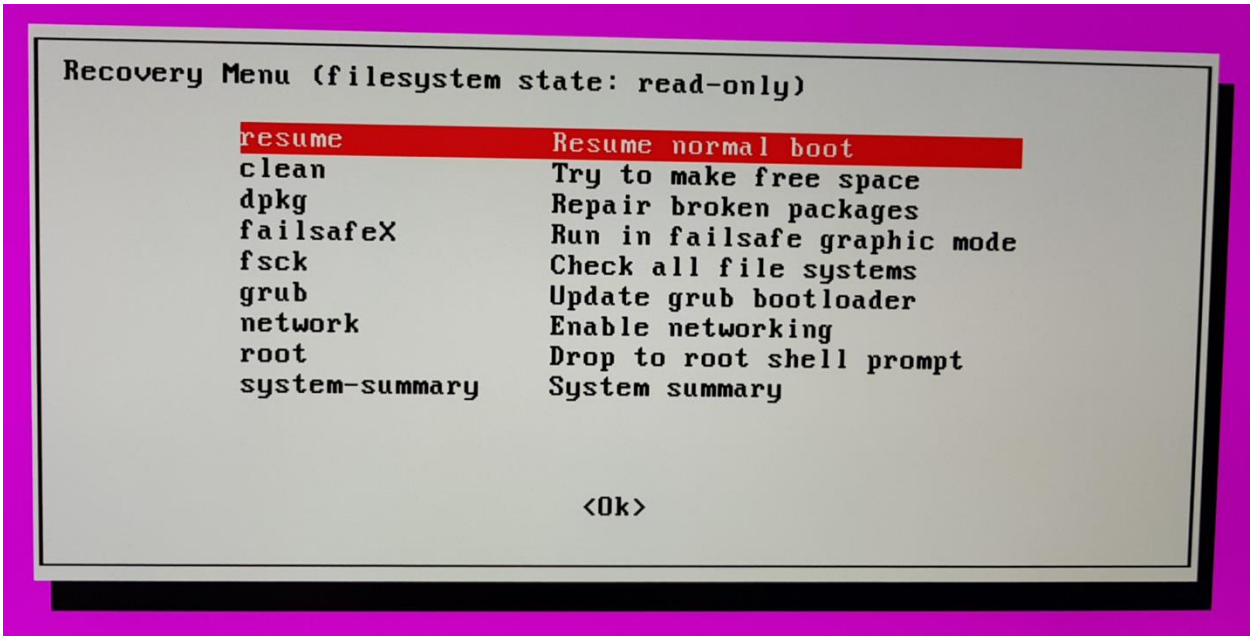

Figure 12: GRUB Resume normal boot

I tried the following commands:

```
cvt 1600 900
```

```
sudo xrandr --newmode "1600x900_60.00" 118.25 1600 1696 1856 2112 900 903 908 934 -hsync +vsync
```

```
sudo xrandr --addmode default "1600x900_60.00"
```

Eventually I was able to enlarge the resolution until 1024x768 XGA resolution, but no way to increase it up to HDTV 1920x1080. Also I was not able to make this configuration permanent: after every reboot VGA resolution was all what I got and I was forced any time to apply the Resume normal boot procedure. I decided to stop here, as the best is to install first the right driver of my NVIDIA Quadro P6000 graphic card.

5.3 Connect target ZCU102 board to Linux host PC with cables

Setting up a serial communication between the Ubuntu Linux PC and the Xilinx Zynq ZCU102 board was not that simple, even assuming https://www.cyberciti.biz/faq/howto-setup-serial-console-on-debian-linux/. I have done a lot of trials, here is the description of what worked fine in my case.

- Run the following Linux commands to understand which ports you have:
  - **dmesg**

```
[ 6449.745350] usb 4-1: USB disconnect, device number 3
[ 9751.915383] usb 3-2: new full-speed USB device number 2 using xhci_hcd
[ 9752.045153] usb 3-2: New USB device found, idVendor=10c4, idProduct=ea71
[ 9752.045160] usb 3-2: New USB device strings: Mfr=1, Product=2, SerialNumber=7
[ 9752.045164] usb 3-2: Product: CP2108 Quad USB to UART Bridge Controller
[ 9752.045167] usb 3-2: Manufacturer: Silicon Labs
```

```
[ 9752.045169] usb 3-2: SerialNumber: BA976282EC0D019911E60126C7C70E6
[ 9752.060869] usbcore: registered new interface driver usbserial
[ 9752.060908] usbcore: registered new interface driver usbserial_generic
[ 9752.060940] usbserial: USB Serial support registered for generic
[ 9752.065397] usbcore: registered new interface driver cp210x
[ 9752.065544] usbserial: USB Serial support registered for cp210x
[ 9752.066674] cp210x 3-2:1.0: cp210x converter detected
[ 9752.066817] usb 3-2: cp210x converter now attached to ttyUSB0
[ 9752.066860] cp210x 3-2:1.1: cp210x converter detected
[ 9752.067016] usb 3-2: cp210x converter now attached to ttyUSB1
[ 9752.067054] cp210x 3-2:1.2: cp210x converter detected
[ 9752.067207] usb 3-2: cp210x converter now attached to ttyUSB2
[ 9752.067243] cp210x 3-2:1.3: cp210x converter detected
[ 9752.067617] usb 3-2: cp210x converter now attached to ttyUSB3
[ 9759.167601] usb 3-1: new high-speed USB device number 3 using xhci_hcd
[ 9759.301175] usb 3-1: New USB device found, idVendor=0403, idProduct=6014
[ 9759.301183] usb 3-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 9759.301186] usb 3-1: Product: Digilent USB Device
[ 9759.301189] usb 3-1: Manufacturer: Digilent
[ 9759.301192] usb 3-1: SerialNumber: 210308A127C5
[ 9752.060869] usbcore: registered new interface driver usbserial
[ 9752.060908] usbcore: registered new interface driver usbserial_generic
[ 9752.060940] usbserial: USB Serial support registered for generic
[ 9752.065397] usbcore: registered new interface driver cp210x
```

- **dmesg | egrep –color 'serial|tty$'**
- **dmesg | egrep –color 'serial|ttyUSB$[01234]'**

- **dmesg | egrep –color 'cp210x'**

```
[ 9752.065544] usbserial: USB Serial support registered for cp210x
[ 9752.066674] cp210x 3-2:1.0: cp210x converter detected
[ 9752.066817] usb 3-2: cp210x converter now attached to ttyUSB0
[ 9752.066860] cp210x 3-2:1.1: cp210x converter detected
[ 9752.067016] usb 3-2: cp210x converter now attached to ttyUSB1
[ 9752.067054] cp210x 3-2:1.2: cp210x converter detected
[ 9752.067207] usb 3-2: cp210x converter now attached to ttyUSB2
[ 9752.067243] cp210x 3-2:1.3: cp210x converter detected
[ 9752.067617] usb 3-2: cp210x converter now attached to ttyUSB3
```

- Install PuTTY with the command
  - **sudo apt-get install putty**
- Once the ZCU102 board is turned on and the two USB cables are connected to the PC, you can execute the following Linux command:
  - **sudo putty /dev/ttyUSB0 –serial –sercfg 115200,8,n,1,N**

  If you want to set PuTTY via its GUI, you need to use **sudo** again

- Once PuTTY terminal pops up you have to configure the eth0 interface of the ZCU102 board:
  - **ifconfig eth0 192.168.1.101 netmask 255.255.255.0**
- You have to connect the target board and the host PC with an Ethernet cable and then set the eth1 host PC interface:
  - **sudo ifconfig eth1 192.168.1.100 netmask 255.255.255.0**
- to check the Ethernet wired communication between the host PC and target board, run the following commands as illustrated in Figure screenshot
  - `ping 192.168.1.100` (from the target board, on the PuTTY terminal)
  - `ping 192.168.1.101` (from host PC, on the Ubuntu terminal)

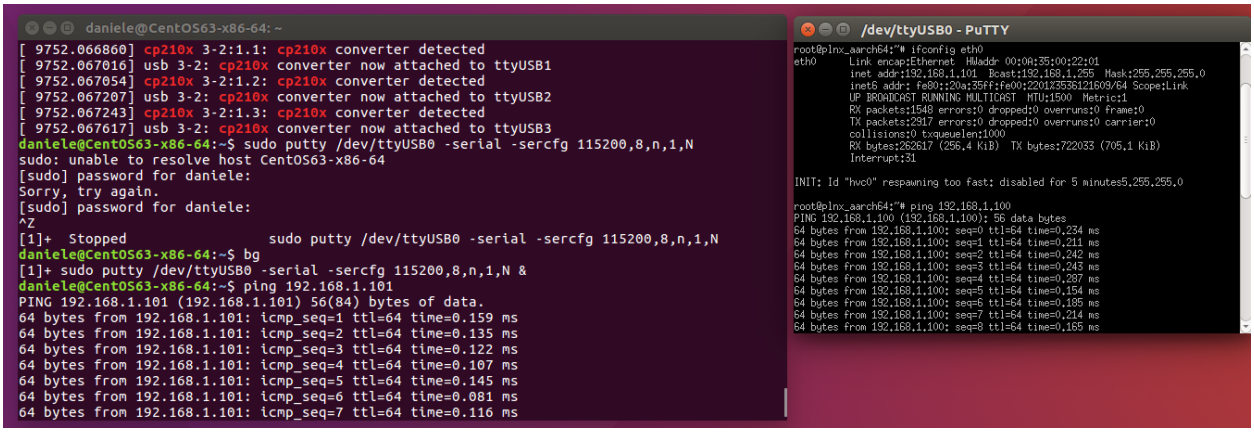

Figure: Ubuntu terminal running on the host (left side) and PuTTY terminal to serially talk with the target board (right side

6.0 Installing NVIDIA Quadro P6000 GPU driver

First I downloaded the display driver of my Quadro P6000 GPU from here:

- http://www.nvidia.com/download/driverResults.aspx/136120/en-us
- https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html
- https://developer.nvidia.com/cuda-toolkit-archive

I then tried to install it in Ubuntu as root, but I stopped it because of a warning related to a mismatch between my GCC compiler release (older) and the one (newer) by which the kernel was compiled.

I decided to stop the above install and I went on with only the following commands.

1) Open a terminal session by hitting key **CTRL+ALT+F2**

2) Stop lightdm (the X-server) with the command: `sudo service lightdm stop`

3) Create a file at **/etc/modprobe.d/blacklist-nouveau.conf** with the following contents:

```
o  blacklist nouveau
o  blacklist lbm-nouveau
o  options nouveau modeset=0
o  alias nouveau off
o  alias lbm-nouveau off
```

Save the file and now execute the following command:

```
echo options nouveau modeset=0 | sudo tee -a
/etc/modprobe.d/nouveau-kms.conf
```

4) **sudo update-initramfs -u**

5) **sudo add-apt-repository ppa:graphics-drivers**

6) **sudo apt-get update**

7) Now install and activate the latest drivers (for me it was version 390)
   sudo apt-get install nvidia-390

8) Reboot your computer.

At the end I could see a desktop finally in 4K resolution, as my ACER 4K monitor.

7.0 Installing NVIDIA CUDA libraries

7.1 CUDA-8.0 ToolKit

1) Perform all the following install instructions:
   - **sudo apt-get update**
   - **sudo apt-get upgrade**
   - **sudo apt-get install build-essential cmake git unzip pkg-config**
   - **sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev**
   - **sudo apt-get install libpng12-dev**
   - **sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev**
   - **sudo apt-get install libv4l-dev**
   - **sudo apt-get install libxvidcore-dev libx264-dev**
   - **sudo apt-get install libgtk-3-dev**
   - **sudo apt-get install libhdf5-serial-dev graphviz**
   - **sudo apt-get install libopenblas-dev libatlas-base-dev gfortran**
   - **sudo apt-get install python-tk python3-tk python-imaging-tk**
   - **sudo apt-get install python2.7-dev python3-dev**
   - **sudo apt-get install linux-image-generic linux-image-extra-virtual**
   - **sudo apt-get install linux-source linux-headers-generic**

2) Before starting to install CUDA I read the material in the following 2 websites:
   https://github.com/bhavykhatri/Installing-_CUDA_toolkit_guide_LINUX/blob/master/README.md
   http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#pre-installation-actions

3) Install the **NVIDIA CUDA Toolkit 8.0**
   2) you need to register first, in order to get then a login and password
   3) surf here: https://developer.nvidia.com/cuda-80-ga2-download-archive
   4) select **Linux** OS, **x86_64** Architecture, **Ubuntu** Distribution, **16.04** Version, **runfile (local)** Installer
   5) you will get the installer (1.5GB) named
      cuda_8.0.61_375.26_linux.run

6) you also need to download a patch named (97.5 MB)
   cuda_8.0.61.2_linux.run
7) Stop the X-server by executing **sudo service lightdm stop**
8) Once download is finished, run the following commands (after changing to the directory where you have placed the file, typically `~/Download`):
   - **chmod +x cuda_8.0.61_375.26_linux.run**
   - **chmod +x cuda_8.0.61.2_linux.run**
   - **mkdir installers**
   - **cd installers**
   - **sudo ../cuda_8.0.61.375.26_linux.run**
   - **sudo ../cuda_8.0.61.2_linux.run**

9) Note that in both cases (both the installer and the patch) you have to accept the NVIDIA EULA license and answer to the following questions:
   - Install the NVIDIA Accelerated Graphic Driver for Linux-x86_64 375.26 (up to you here: if you do it, remember that the X-server must be off)
   - Install OpenGL Libraries (NO)
   - Install the CUDA 8.0 Toolkit (YES)
   - Run `nvidia-xconfig` (YES)
   - Install the CUDA 8.0 Samples (YES)
   - set the CUDA Toolkit Install directory (accept default: `/usr/local/cuda-8.0`)
10) If you get an error, you have to look at the log file `/var/log/nvidia-installer.log`. In my case I missed the libraries `libGLU.so libXmu.so`, as illustrated in Figure 13; after a look at here https://stackoverflow.com/questions/22360771/missing-recommended-library-libglu-so I installed the missing libraries with the command
    apt-get install libglu1-mesa libxi-dev libxmu-dev libglu1-mesa-dev

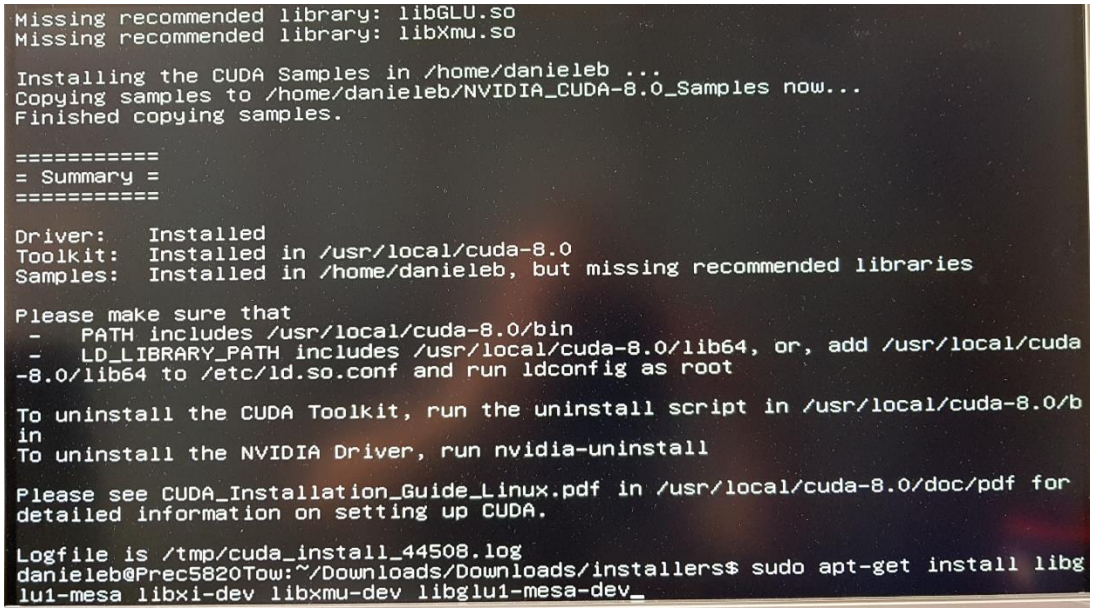

Figure 13: results of first trial to install CUDA 8.0

11) When things go well you should see something like the screenshot of Figure 14:

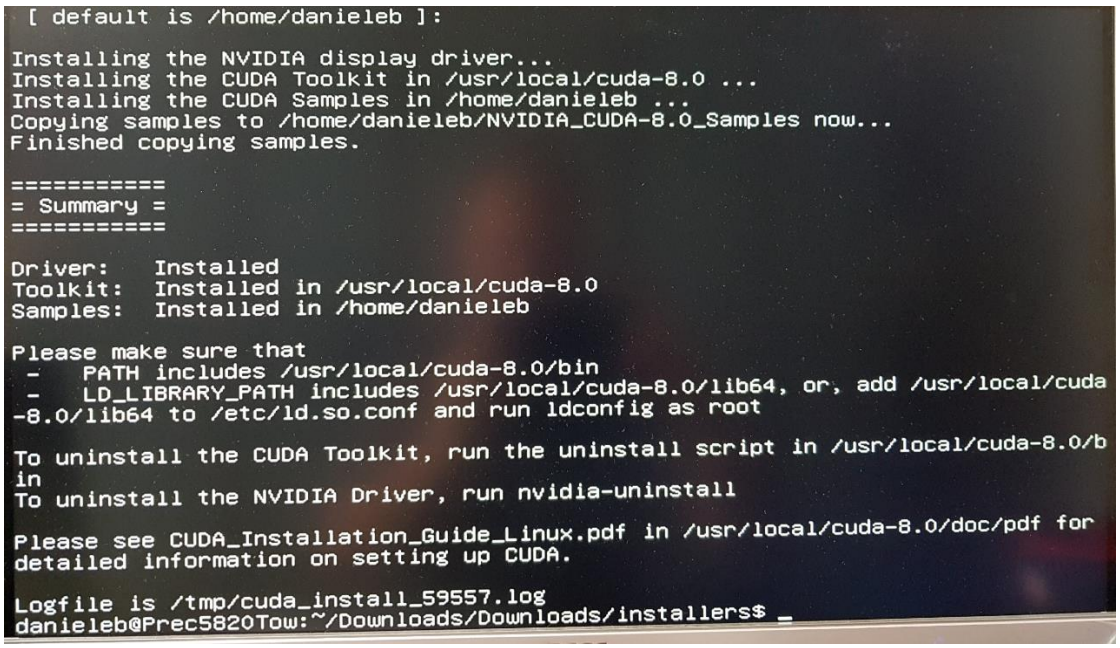

Figure 14: CUDA-8.0 correct install result

12) Add the NVIDIA loadable kernel module (KLM) to the Linux Kernel:
    - **sudo modprobe nvidia**

13) Add the following lines to **~/.bashrc** file
    a. # NVIDIA CUDA Toolkit
    b. export PATH=/usr/local/cuda-8.0/bin:$PATH
    c. export LD_LIBRARY_PATH=/usr/local/cuda-8.0/lib64/

14) Now execute test the CUDA Toolkit installation by compiling the deviceQuery example program and running it:
    d. source ~/.bashrc
    e. cd /usr/local/cuda-8.0/samples/1_Utilities/deviceQuery
    f. sudo make
    g. ./deviceQuery

In my case, unfortunately I got an error and I had to look at here:

https://devtalk.nvidia.com/default/topic/760872/ubuntu-12-04-error-cudagetdevicecount-returned-30/

but the situation become even worst: at any reboot the X-server was not restarting and I could only use the terminal (CTRL+ALT+F1). Therefore I decided to install the driver of Section **6.0 Installing NVIDIA Quadro P6000 GPU driver** with the command

sudo ./NVIDIA-Linux-x86_64-390.77.run

Again the installer was complaining of not having the right GCC release, but I answered YES to all questions and I installed it anyway. After the reboot I could finally see again my desktop in 4K

resolution and I was able to run the commands of item 14 in the above list, now the output looks correct, as illustrated in Figure 15

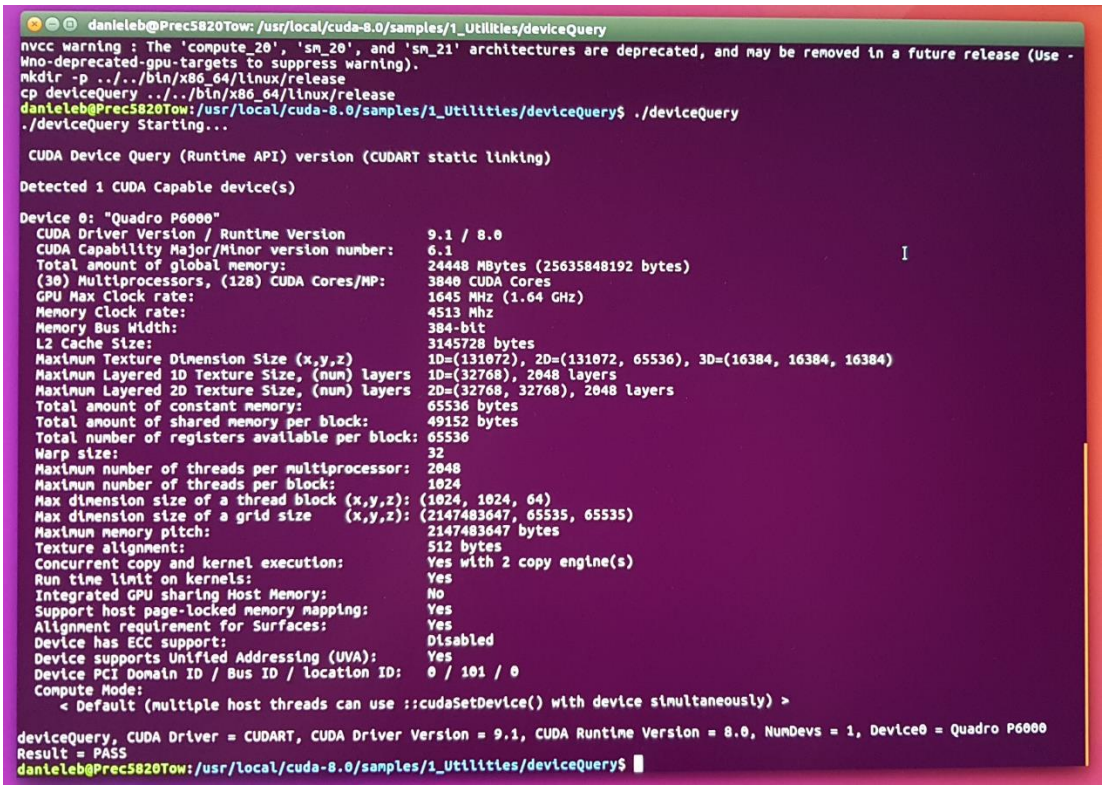

Figure 15: output of NVIDIA GPU Device Query

7.2 cuDNN 7.0.5

Download **NVIDIA cuDNN v7.0.5** (you need to use your login and password) for Deep Learning (DL), from these various URLs, listed in order:

- https://developer.nvidia.com/deep-learning-software (click on DL Primitives **cuDNN**)
- https://developer.nvidia.com/cudnn (click on **Download** and then **Agree** on Licensing terms)
- https://developer.nvidia.com/rdp/cudnn-download
- select **Dowload cuDNN v7.0.5 (Dec 5, 2017) for CUDA 8.0**
- select cuDNN v7.0.5 Library for Linux
- you will get the file (268 MB) named **cudnn-8.0-linux-x64-v7.tgz**
- put such file in the `~/Download/installers` directory, then execute the following commands:

```
a. cd ~/Download/installers
b. tar -zxf cudnn-8.0-linux-x64-v7.tgz
c. cd cuda
d. sudo cp -P ./lib64/*   /usr/local/cuda-8.0/lib64/
e. sudo cp -P ./include/* /usr/local/cuda-8.0/include/
f. cd ~
```

7.3 NCCL v1.2.3

DeePhi DNNDK tool requires NCCL v1.2.3. Download "NCCL v1.2.3 (Linked with CUDA 8.0)" from here: https://github.com/NVIDIA/nccl/archive/v1.2.3-1+cuda8.0.tar.gz

Put such file into your Download area and then launch the following commands (see below screenshot)

```
a. cd ~/Downloads/
b. tar -xvf nccl-1.2.3-1-cuda8.0.tar.gz
c. cd nccl-1.2.3-1-cuda8.0
d. sudo make install –j
e. sudo ldconfig /usr/local/cuda/lib64
f. make CUDA_HOME=/usr/local/cuda test
```

```
root2@Prec5820Tow:~/Downloads$ cd nccl-1.2.3-1-cuda8.0/
root2@Prec5820Tow:~/Downloads/nccl-1.2.3-1-cuda8.0$ sudo make install -j
removed '/usr/local/lib/libnccl.so'
'build/lib/libnccl.so' -> '/usr/local/lib/libnccl.so'
removed '/usr/local/lib/libnccl.so.1'
'build/lib/libnccl.so.1' -> '/usr/local/lib/libnccl.so.1'
'build/lib/libnccl.so.1.2.3' -> '/usr/local/lib/libnccl.so.1.2.3'
'build/include/nccl.h' -> '/usr/local/include/nccl.h'
root2@Prec5820Tow:~/Downloads/nccl-1.2.3-1-cuda8.0$ sudo ldconfig /usr/local/cuda/lib64
root2@Prec5820Tow:~/Downloads/nccl-1.2.3-1-cuda8.0$ sudo make CUDA_HOME=/usr/local/cuda test
root2@Prec5820Tow:~/Downloads/nccl-1.2.3-1-cuda8.0$ ./build/test/single/all_reduce_test  100000000
# Using devices
#   Rank  0 uses device  0 [0x65] Quadro P6000

#                                                     out-of-place                       in-place
#      bytes             N    type      op     time  algbw  busbw      res     time  algbw  busbw      res
   100000000     100000000    char     sum    0.526  190.11   0.00    0e+00    0.007  13627.69   0.00    0e+00
   100000000     100000000    char    prod    0.519  192.52   0.00    0e+00    0.007  15332.72   0.00    0e+00
   100000000     100000000    char     max    0.519  192.86   0.00    0e+00    0.007  14541.22   0.00    0e+00
   100000000     100000000    char     min    0.524  190.96   0.00    0e+00    0.007  14839.00   0.00    0e+00
   100000000      25000000     int     sum    0.528  189.28   0.00    0e+00    0.006  15578.75   0.00    0e+00
   100000000      25000000     int    prod    0.528  189.47   0.00    0e+00    0.007  14740.57   0.00    0e+00
   100000000      25000000     int     max    0.524  190.85   0.00    0e+00    0.007  14564.52   0.00    0e+00
   100000000      25000000     int     min    0.531  188.49   0.00    0e+00    0.007  14974.54   0.00    0e+00
   100000000      50000000    half     sum    0.527  189.70   0.00    0e+00    0.007  15246.23   0.00    0e+00
   100000000      50000000    half    prod    0.523  191.12   0.00    0e+00    0.007  14283.67   0.00    0e+00
   100000000      50000000    half     max    0.527  189.58   0.00    0e+00    0.007  14889.82   0.00    0e+00
   100000000      50000000    half     min    0.527  189.76   0.00    0e+00    0.006  15537.60   0.00    0e+00
   100000000      25000000   float     sum    0.528  189.28   0.00    0e+00    0.006  16048.79   0.00    0e+00
   100000000      25000000   float    prod    0.528  189.51   0.00    0e+00    0.007  14883.17   0.00    0e+00
   100000000      25000000   float     max    0.531  188.39   0.00    0e+00    0.007  14695.08   0.00    0e+00
   100000000      25000000   float     min    0.528  189.29   0.00    0e+00    0.006  16199.58   0.00    0e+00
   100000000      12500000  double     sum    0.529  189.06   0.00    0e+00    0.007  14509.58   0.00    0e+00
   100000000      12500000  double    prod    0.527  189.92   0.00    0e+00    0.006  16302.58   0.00    0e+00
   100000000      12500000  double     max    0.529  188.99   0.00    0e+00    0.006  16608.54   0.00    0e+00
   100000000      12500000  double     min    0.523  191.29   0.00    0e+00    0.006  15642.11   0.00    0e+00
   100000000      12500000   int64     sum    0.528  189.52   0.00    0e+00    0.006  15765.41   0.00    0e+00
   100000000      12500000   int64    prod    0.524  190.71   0.00    0e+00    0.007  14925.37   0.00    0e+00
   100000000      12500000   int64     max    0.529  189.02   0.00    0e+00    0.007  14740.57   0.00    0e+00
   100000000      12500000   int64     min    0.528  189.46   0.00    0e+00    0.007  14889.82   0.00    0e+00
   100000000      12500000  uint64     sum    0.525  190.30   0.00    0e+00    0.006  15936.25   0.00    0e+00
   100000000      12500000  uint64    prod    0.528  189.50   0.00    0e+00    0.007  14999.25   0.00    0e+00
   100000000      12500000  uint64     max    0.527  189.66   0.00    0e+00    0.007  15316.28   0.00    0e+00
   100000000      12500000  uint64     min    0.526  190.03   0.00    0e+00    0.006  15436.86   0.00    0e+00

root2@Prec5820Tow:~/Downloads/nccl-1.2.3-1-cuda8.0$
```

Figure: screenshot of how installing nccl1.0

then follow the instructions available on the **README.md** file to test it.

7.4 Optional: CUDA-9.1 ToolKit

Instructions to alternatively install NVIDIA CUDA ToolKit 9.1 is the same as the 8.0, just go to the following pages:

- https://developer.nvidia.com/cuda-toolkit-archive
- https://developer.nvidia.com/cuda-91-download-archive
- https://developer.nvidia.com/cuda-91-download-archive?target_os=Linux&target_arch=x86_64&target_distro=Ubuntu&target_version=1604&target_type=runfilelocal

Alternatively to my instructions, you might want to use the following instructions (by Jon Cory), which I have not checked:

1. Install repository meta-data (as well as any patches)

```
$ sudo dpkg -i cuda-repo-<distro>_<version>_<architecture>.deb
```

2. Installing the CUDA public GPG key

   When installing using the local repo:

```
$ sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub

os/<distro>/<architecture>/7fa2af80.pub
```

3. Update the Apt repository cache

```
$ sudo apt-get update
```

4. Install CUDA

```
$ sudo apt-get install cuda
```

5. The PATH variable needs to include `/usr/local/cuda-9.1/bin` with the command
   `$ export PATH=/usr/local/cuda-9.1/bin${PATH:+:${PATH}}`
   In addition, when using the runfile installation method, the LD_LIBRARY_PATH variable needs to contain /usr/local/cuda-9.1/lib64 on a 64-bit system, or /usr/local/cuda-9.1/lib on a 32-bit system
6. To change the environment variables for 64-bit operating systems:
   `$ export LD_LIBRARY_PATH=/usr/local/cuda-9.1/lib64\`
   `${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}`

7.5 Optional: cuDNN 7.0.5 for CUDA toolkit 9.1 (by J. Cory, also checked by me)

Instructions are taken from here:

http://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html

In order to download cuDNN, ensure you are registered for the NVIDIA Developer Program.

1. Go to: NVIDIA cuDNN home page.
2. Click **Download**.
3. Complete the short survey and click **Submit**.
4. Accept the Terms and Conditions. A list of available download versions of cuDNN displays.
5. Select the cuDNN version you want to install (I used the Debian Packages). A list of available resources displays.
6. Navigate to your <cudnnpath> directory containing cuDNN Debian file.
7. Install the runtime library, for example:

```
sudo dpkg -i libcudnn7_7.0.3.11-1+cuda9.0_amd64.deb
```

8. Install the developer library, for example:

```
sudo dpkg -i libcudnn7-dev_7.0.3.11-1+cuda9.0_amd64.deb
```

9. Install the code samples and the cuDNN Library User Guide, for example:

```
sudo dpkg -i libcudnn7-doc_7.0.3.11-1+cuda9.0_amd64.deb
```

10. Copy the cuDNN sample to a writable path.

```
$cp -r /usr/src/cudnn_samples_v7/ $HOME
```

11. Go to the writable path.

```
$ cd  $HOME/cudnn_samples_v7/mnistCUDNN
```

12. Compile the mnistCUDNN sample.

```
$make clean && make
```

13. Run the mnistCUDNN sample.

```
$ ./mnistCUDNN
```

If cuDNN is properly installed and running on your Linux system, you will see a message similar to the following:

```
Test passed!
```

8.0 Create your Python2.7 Virtual Env. with ML-related packages

Whatever here described is also valid in general for Python3.5, with really few differences.

- Perform all the following install instructions:
  - `wget https://bootstrap.pypa.io/get-pip.py`
  - `sudo python get-pip.py`
  - `sudo python3 get-pip.py`
  - `sudo pip2 install virtualenv virtualenvwrapper`
  - `sudo rm -rf ~/.cache/pip get-pip.py`
- Add the following lines to the **`~/.bashrc`** file
  - `# virtualenv and virtualenvwrapper`
  - `export WORKON HOME=$HOME/.virtualenvs`
  - `export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python2`
  - `source /usr/local/bin/virtualenvwrapper.sh`

  In case of Python3 the third line would obviously be:
  - `export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3`

- Now reload the changes by running:
  - `source ~/.bashrc`

  WARNING: DO NOT MIX UP ENVIRONMENTS WITH PYTHON 2 or 3.
  Always "**`source ~/.bashrc`**" with the right python version you want to use (for example Python2) by commenting the lines related to the other python version not used (for example Python3), otherwise you will have many errors.

- From now until the end of this document I will use only Python2.7 (shortly Python2), as most of the Caffe tutorials on Internet use Python2.7, note that the commands are basically the same for both Python 2.7 and 3.5 releases.

- Create a Python2, based virtual environment, for example "caffe_py27":
  - `mkvirtualenv caffe_py27 -p python2`
  - `workon caffe_py27`

  In case of python3 the first line would be
  - `mkvirtualenv caffe_py35 -p python3`

- Import all the following ML related packages (which I have already checked to work fine in my Ubuntu 16.04 LapTop PC and saved with the command **`pip2 freeze > filename.txt`**) by running the command (note that you must be inside the virtualenv)
  - **`pip2 install -r ~/scripts/caffe_py27_requirements.txt`**

8.1 Content of caffe_py27_requirements.txt file

```
absl-py==0.1.13
astor==0.6.2
backports.functools-lru-cache==1.5
backports.shutil-get-terminal-size==1.0.0
```

```
backports.weakref==1.0.post1
bleach==1.5.0
certifi==2018.1.18
chardet==3.0.4
cloudpickle==0.5.3
cycler==0.10.0
Cython==0.28.4
dask==0.18.2
decorator==4.3.0
enum34==1.1.6
funcsigs==1.0.2
futures==3.2.0
gast==0.2.0
graphviz==0.8.4
grpcio==1.10.1
h5py==2.8.0
html5lib==0.9999999
idna==2.6
imutils==0.4.6
ipython==5.7.0
ipython-genutils==0.2.0
Keras==2.1.5
kiwisolver==1.0.1
leveldb==0.194
lmdb==0.94
Markdown==2.6.11
matplotlib==2.2.2
mock==2.0.0
networkx==2.1
nose==1.3.7
numpy==1.15.0
olefile==0.44
pandas==0.23.3
pathlib2==2.3.2
pbr==4.2.0
pexpect==4.6.0
pickleshare==0.7.4
Pillow==5.2.0
progressbar2==3.37.0
prompt-toolkit==1.0.15
protobuf==3.6.0
ptyprocess==0.6.0
pydot==1.2.4
Pygments==2.2.0
pyparsing==2.2.0
python-dateutil==2.7.3
python-gflags==3.1.2
python-utils==2.3.0
pytz==2018.5
PyWavelets==0.5.2
PyYAML==3.13
requests==2.18.4
scandir==1.7
```

```
scikit-image==0.14.0
scikit-learn==0.19.1
scipy==1.0.0
simplegeneric==0.8.1
six==1.11.0
stevedore==1.29.0
subprocess32==3.5.2
tensorboard==1.7.0
tensorflow-gpu==1.4.1
tensorflow-tensorboard==0.4.0
termcolor==1.1.0
toolz==0.9.0
traitlets==4.3.2
urllib3==1.22
virtualenv==16.0.0
virtualenv-clone==0.3.0
virtualenvwrapper==4.8.2
wcwidth==0.1.7
Werkzeug==0.14.1
```

8.2 Install other packages needed by Caffe, included openCV 2.4

Follow the next instructions out of any python virtualenv:

```
o  sudo apt-get update
o  sudo apt-get upgrade
o  sudo apt-get install -y libleveldb-dev libsnappy-dev protobuf-compiler
o  sudo apt-get install -y --no-install-recommends libboost-all-dev
o  sudo apt-get install -y libgflags-dev libgoogle-glog-dev liblmdb-dev
o  sudo apt-get install -y python-pip
o  sudo apt-get install -y python-numpy python-scipy
o  sudo apt-get install -y python-opencv
o  sudo apt-get install -y python-lmdb
```

Note that the below packages should be already installed from previous Sections

```
o  sudo apt-get install -y build-essential cmake git pkg-config
o  sudo apt-get install -y libprotobuf-dev libhdf5-serial-dev
o  sudo apt-get install -y libatlas-base-dev
o  sudo apt-get install -y python-dev
```

For OpenCV 2.4 you only need to run the below command:

```
o  sudo apt-get install -y libopencv-dev
```

Note that the cv2 package was installed in **/usr/lib/python2.7/dist-packages** instead of **/usr/local/lib/python2.7/dist-packages**, therefore I applied the following workaround:

```
o  cd ~/.virtualenvs/caffe_py27/local/lib/python2.7/site-packages/
o  ln -s /usr/lib/python2.7/dist-packages/cv2.x86_64-linux-gnu.so cv2.so
o  cd ~
```

Now test your OpenCV install, whatever python you are using

```
python
import cv2
cv2.__version__
cv2.__file__
quit()
```

In case cv2 would not be import (for python2 for example), just add the following 2 lines just before the "import cv2":

```
python
import sys
sys.path.append("/usr/local/lib/python2.7/site-packages")
import cv2
cv2.__version__
quit()
```

Test your TensorFlow install:

```
python
import tensorflow
tensorflow.__version__
quit()
```

If you have errors after <import panda> you might need to re-install panda with the following command (at this point it is better you also re-install tensorflow):

```
sudo pip2 install pandas
```

Test your Keras install:

```
python
import keras
keras.__version__
quit()
```

Alternatively, in case you want to use OpenCV 3.x, you also need to install all the below packages (again, some of them are redundant as already installed previously):

```
sudo apt-get install --assume-yes build-essential cmake git

sudo apt-get install --assume-yes pkg-config unzip ffmpeg qtbase5-dev
python-dev python3-dev python-numpy python3-numpy

sudo apt-get install --assume-yes libopencv-dev libgtk-3-dev libdc1394-22
libdc1394-22-dev libjpeg-dev libpng12-dev libtiff5-dev libjasper-dev

sudo apt-get install --assume-yes libavcodec-dev libavformat-dev
libswscale-dev libxine2-dev libgstreamer0.10-dev libgstreamer-plugins-
base0.10-dev

sudo apt-get install --assume-yes libv4l-dev libtbb-dev libfaac-dev
libmp3lame-dev libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev

sudo apt-get install --assume-yes libvorbis-dev libxvidcore-dev v4l-utils
python-vtk

sudo apt-get install --assume-yes liblapacke-dev libopenblas-dev
checkinstall
sudo apt-get install --assume-yes libgdal-dev
```

8.3 How to install openCV 3.3 from scratch (optional)

If you want to install OpenCV 3.3 instead of 2.4 look at here.

1) Compile and install OpenCV 3.3 (also 3.2 or 3.1 are fine) without CUDA support (simply to avoid many compilation errors due to CUDA, all in all I want the DL stuff to be CUDA accelerated, not the OpenCV stuff)

```
o  cd ~
o  wget -O opencv.zip
   https://github.com/Itseez/opencv/archive/3.3.0.zip
o  wget -O opencv_contrib.zip
   https://github.com/Itseez/opencv_contrib/archive/3.3.0.zip
o  mkdir opencv
o  mv opencv_contrib.zip opencv.zip ./opencv
o  cd opencv
o  unzip opencv.zip
o  unzip opencv_contrib.zip
o  cd ./opencv-3.3.0/
o  mkdir build
o  cd build
o  cmake -D CMAKE_BUILD_TYPE=RELEASE \
o  -D CMAKE_INSTALL_PREFIX=/usr/local \
o  -D WITH_CUDA=OFF \
o  -D INSTALL_PYTHON_EXAMPLES=ON \
o  -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
o  -D BUILD_EXAMPLES=ON ..
```

2) in case of errors you might need to add also the following flags (before any cmake trial, create a completely new empty `build` directory) to the previous ones:

```
o  -D PYTHON_EXECUTABLE=~/.virtualenvs/cudadnn/bin/python \
o  -D ENABLE_PRECOMPILED_HEADERS=OFF \
o  -D CUDA_GENERATION=AUTO \
```

3) Now compile OpenCV

```
o  make -j4
o  sudo make install
o  sudo /bin/bash -c 'echo "/usr/local/lib" >
   /etc/ld.so.conf.d/opencv.conf'
o  sudo ldconfig
o  sudo apt-get update
o  sudo apt-get install checkinstall
o  sudo checkinstall
o  cd ~
```

4) Reboot your machine. You are now ready to install Caffe with OpenCV3.x

8.4 File install_caffe_py27_packages.sh

I have captured all the **`sudo apt-get install`** commands of SubSections 7.1 and 8.2 in a file to be executed with the command **`source install_caffe_py27_packages.sh`**

See below its content:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install build-essential cmake git unzip pkg-config
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx624-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install libgtk-3-dev
sudo apt-get install libhdf5-serial-dev graphviz
sudo apt-get install libopenblas-dev libatlas-base-dev gfortran
sudo apt-get install python2.7-dev python3-dev
sudo apt-get install linux-image-generic linux-image-extra-virtual
sudo apt-get install linux-source linux-headers-generic
sudo apt-get install libglu1-mesa libxi-dev libxmu-dev libglu1-mesa-dev
sudo apt-get install --no-install-recommends libboost-all-dev
sudo apt-get install -y build-essential
sudo apt-get install -y cmake
sudo apt-get install -y git
sudo apt-get install -y pkg-config
sudo apt-get install -y libprotobuf-dev
sudo apt-get install -y libleveldb-dev
sudo apt-get install -y libsnappy-dev
sudo apt-get install -y libhdf5-serial-dev
sudo apt-get install -y protobuf-compiler
sudo apt-get install -y libatlas-base-dev
sudo apt-get install -y libgflags-dev
sudo apt-get install -y libgoogle-glog-dev
sudo apt-get install -y liblmdb-dev
sudo apt-get install python-pip
sudo apt-get install python-dev
sudo apt-get install python-numpy
sudo apt-get install python-scipy
sudo apt-get install python-opencv
sudo apt-get install python-lmdb
sudo apt-get install libopencv-dev
```

9.0 Install Caffe BVLC 1.0

First of all you need to download the code. **Caffe BVLC 1.0** is public available here: https://github.com/BVLC/caffe. Alternatively you could use one of these two forks:

- **Ristretto.** It was developed by GL-Research and it is available here: https://github.com/pmgysel/caffe. There is a WiKi page associated to it: https://gl-research.com/caffe/ristretto/wikis/home and you need to contact the owner to get access to it (Javier Garcia, email address: jgarcia@gl-research.com).

- **Caffe-SSD-Ristretto** is Xilinx internal only available here: https://gitenterprise.xilinx.com/smenon/Caffe-SSD-Ristretto

They both contain the original Caffe BVLC 1.0 plus other additions. In particular, Caffe-SSD-Ristretto contains also Ristretto and some changes to Caffe BVLC 1.0 suitable to manage the SSD Object Detection. Installing the Ristretto or Caffe-SSD-Ristretto is the same procedure (I have both of them in my laptop), therefore I will describe only the Caffe-BVLC1v0, being it more general.

Put in your `~/.bashrc` file the following line, by adjusting the pathname where you files stay

```
export CAFFE_ROOT=/home/danieleb/caffe_tools/BVCL1v0-Caffe
```

```
1) workon caffe_py27
2) cd $CAFFE_ROOT
3) cp Makefile.config.example Makefile.config
```

4) Now edit **Makefile.config** before running any other step.
   - Uncommon the `CPU_ONLY:=1` line at the top, if you don't have a GPU. On the contrary, if you have a GPU, leave it commented and uncomment `USE_CUDNN:=1`
   - `USE_OPENCV:=1`, to enable OpenCV .
   - Remove the following 2 lines
     ```
     -gencode arch=compute_20,code=sm_20 \
     -gencode arch=compute_20,code=sm_21 \
     ```
   - Check your PYTHON_INCLUDE to be similar to this (add the third line only if your python2.7 previous installation generated files in both `/usr/lib` and `/usr/local/lib`, otherwise leave it commented)
     ```
     PYTHON_INCLUDE := /usr/include/python2.7 \
           /usr/lib/python2.7/dist-packages/numpy/core/include
           /usr/local/lib/python2.7/dist-packages/numpy/core/include
     ```
   - Add the following two lines to **Makefile.config** (make sure each line stays in a single line, without any carriage return), otherwise you will get a compilation error during the step of **`make all`**
     ```
     o INCLUDE_DIRS:=$(PYTHON_INCLUDE) /usr/local/include
       /usr/include/hdf5/serial
     o LIBRARY_DIRS:=$(PYTHON_LIB) /usr/local/lib /usr/lib
       /usr/lib/x86_64-linux-gnu /usr/lib/x86_64-linux-
       gnu/hdf5/serial  /usr/local/share/OpenCV/3rdpar ty/lib
     ```

5) Now you should edit also the **Makefile**, to avoid possible future compilation errors that otherwise could compare. You can first do a trial without modifying it and then, if you got errors, try the below changes:

- Replace the following line
  ```
  NVCCFLAGS+= -ccbin=$(CXX) -Xcompiler -fPIC $(COMMON_FLAGS)
  ```

  with the following line (everything on the same line, no carriage-return)

  ```
  NVCCFLAGS+=-D_FORCE_INLINES -ccbin=$(CXX) -Xcompiler -fPIC
  $(COMMON_FLAGS)
  ```

- if you are using OpenCV 3.x and not OpenCV2, you also need to add to the following LIBRARIES
  ```
  opencv_imgcodecs opencv_videoio
  ```

- Now open the file **CMakeLists.txt** and add the following two lines:
  ```
  # ---[ Includes
  set(${CMAKE_CXX_FLAGS} "-D_FORCE_INLINES ${CMAKE_CXX_FLAGS}")
  ```

6) Now you can run the following commands to build and test Caffe-Ristretto (note that 8 is the amount of parallel CPUs of your PC, if you have less you need to use a small number):

```
o  cd /usr/lib/x86_64-linux-gnu
o  sudo ln -s libhdf5_serial.so.10.0.2 libhdf5.so
o  sudo ln -s libhdf5_serial_hl.so.10.0.2 libhdf5_hl.so
o  cd $CAFFE_ROOT
o  make clean
o  make all -j 8
o  make test -j 8
o  make runtest -j 8
o  make pycaffe
o  make distribute
```

7) Finally execute the following commands (or put them into your **~/.bashrc** Linux file)

```
o  export
   LD_LIBRARY_PATH=$CAFFE_ROOT/distribute/lib:$LD_LIBRARY_PATH
o  export PYTHONPATH=$CAFFE_ROOT/distribute/python:$PYTHONPATH
```

9.1 Script to activate the virtual environment

My solution to have parallel different virtual environments in a clean way is based on a proper **.bashrc** file and a shell script (in **/home/danieleb/scripts**), each one for a specific virtualenv.

In this case I have the **activate_bvlc1v0_py27.sh** using **.bashrc** to be launched every time you want to enter in this virtualenv (similarly to what done in AWS by Amazon) with the command

source activate_bvcl1v0_py27.

Here is the content of the list lines I added to the original **.bashrc**

```
# MACHINE LEARNING: reset variables
LD_LIBRARY_PATH=$(getconf LD_LIBRARY_PATH)
PYTHONPATH=$(getconf PYTHONPATH)
CAFFE_ROOT=$(getconf CAFFE_ROOT)

#NVIDIA CUDA 8.0 Toolkit
export PATH=/usr/local/cuda-8.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-8.0/lib64/:$LD_LIBRARY_PATH
```

and here is the content of **activate_bvlc1v0_py27.sh**

```
echo ""
echo "VIRTUALENV WITH BVLC 1v0 Caffe, PYTHON2.7, CUDA-8.0 cuDNN-7.0.5,
TENSORFLOW-GPU 1.4.1"
echo ""
source ~/.bashrc

# caffe related variables
export CAFFE_ROOT=/home/danieleb/caffe_tools/BVLC1v0-Caffe
export LD_LIBRARY_PATH=$CAFFE_ROOT/distribute/lib
export PYTHONPATH=$CAFFE_ROOT/distribute/python

# virtualenv for python2.7
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python
export VIRTUALENVWRAPPER_VIRTUALENV=/usr/local/bin/virtualenv
source /usr/local/bin/virtualenvwrapper.sh
export PYTHONPATH=/usr/local/lib/python2.7/dist-packages:$PYTHONPATH

workon caffe_py27
```

9.2 Possible Caffe build errors

a) See the compilation problem I had on the file **cudnn.hpp** which I had to correct as illustrated in Figure 16, based on https://github.com/MhLiao/TextBoxes_plusplus/pull/6/files?diff=split

b) If you used Python3 instead of Python2 you will probably have a linker error about a missing library named **libboost_python3.so** after the first **make all** trial. If you look at the directory

```
o  ls -l /usr/lib/x86_64-linux-gnu/libboost_python*
```

you will find something like `libboost_python-py35.so.1.58.0`

therefore, create a softlink with the commands

```
o  cd /usr/lib/x86_64-linux-gnu
o  ln -s libboost_python-py35.so.1.58.0 libboost_python3.so
```

c) If you get an error during the `make runtest -j8`, about "**LRN layer only supports minifloat**" therefore you need to add the following lines before line 46 of file `src/caffe/test/test_layer_factory.cpp`

```
if (iter->first ==”LRNRistretto”) {
   layer_param.mutable_quantization_param()->set_precision(
   caffe::QuatizationParameter_Precision_MINIFLOAT):
}
```

d) If you compiled with `GPU (USE_CUDNN:= 1`, in Makefile.config) you might need to modify line 112 of file `include/caffe/util/cudnn.hpp`
by replacing
`cudnnSetConvolution2Descriptor(*conv,`
with
`cudnnSetConvolution2Descriptor_v4(*conv`,
Note that you just added `"_v4"` to the function name.
Note that I am not sure anymore about this point. It happened to me in the first Ristretto install on my LapTop but never happened now with the Dell 5820 Desktop

e) Due to some runtime errors in the test of some python scripts I had to launch the two following commands to solve them, from caffe_py27 virtual environment launch:

```
o  pip2 install lmdb
o  sudo apt-get install python-tk
```

9.3 Two Very Recent Errors

Recently I have found two new errors that never appeared months ago.

The first one is described here: https://github.com/BVLC/caffe/issues/6550 and it is related to "**H5Fis_hdf5"** error probably due to some wrong merge of different Caffe releases. I therefore have added the original code that was working fine for me and replace the newest one.

The second is related to "**Importerror: no module named google.protobuf"** and does not depend on Caffe, but more on Google. It is described here:

https://stackoverflow.com/questions/38680593/importerror-no-module-named-google-protobuf

Also in this case I have added the original python package **google** to be copied in the correct directory of the virtual environment (assumed to have name **`bvlc1v0_py27`**), as shown in the below commands:

```
o  mkdir tmp
o  cp scripts/google_patch.tar.gz tmp
o  cd tmp
```

```
o  tar -xvf google_patch.tar.gz
o  cp -r google
   ~/.virtualenvs/bvlc1v0_py27/local/lib/python2.7/site-packages/
o  sudo touch  ~/.virtualenvs/bvlc1v0_py27/lib/python2.7/site-
   packages/google/__init__.py
```

If you get those two errors I think it is better you go directly to Section 10 where I have put these solutions in the script named **`install_caffe.sh`**.

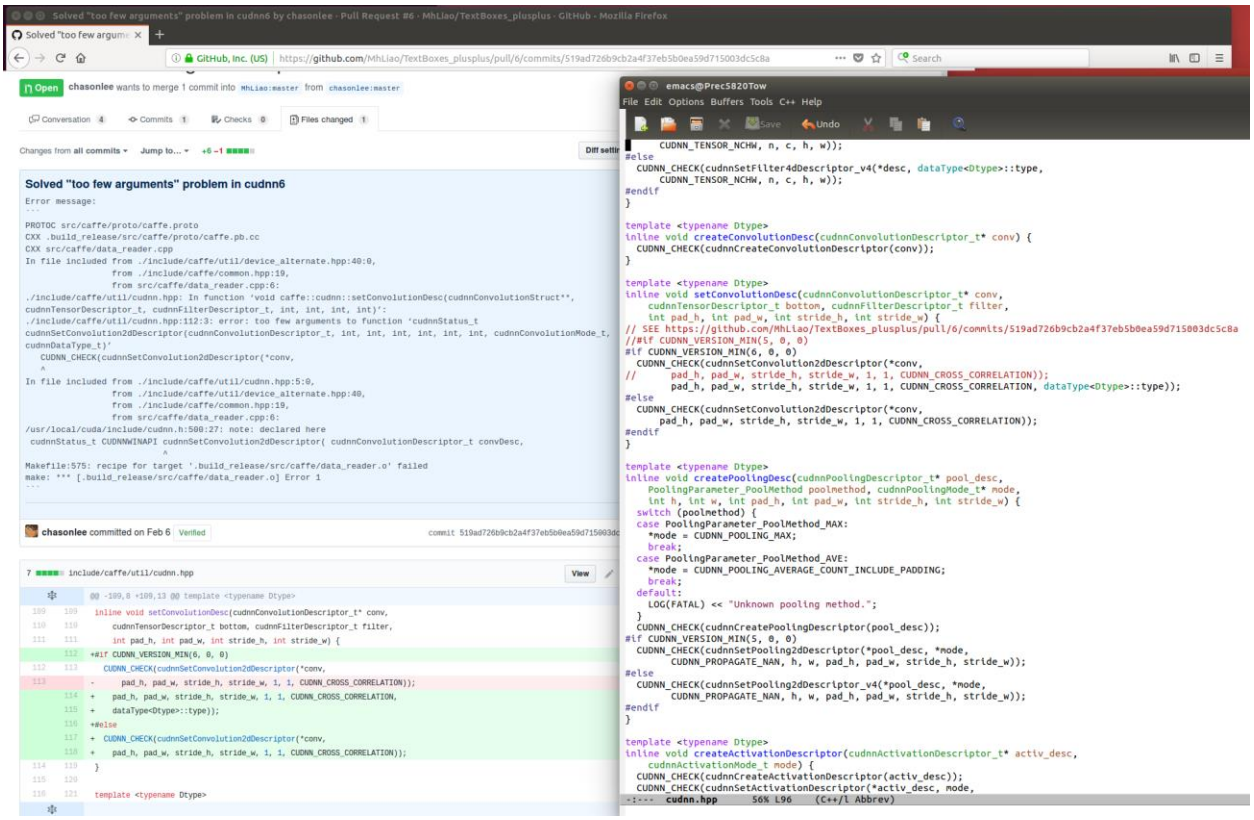

Figure 16: solving a compilation problem due to ***`cudnn.hpp`***

10.0 Shortcut for last two sections

Running all the instructions of Sections 8 and 9 can be really painful. In an attempt to make your life easier I have developed a shortcut solution that should hopefully execute everything automatically for you.

The archive **`install_caffe_scripts.tar`** contains everything you need: Makefiles, .bashrc and activation shell scripts. By launching those scripts everything should be automatically installed, provided you did not fail in installing NVIDIA GPU driver and CUDA 8.0 libraries as explained in Sections 6 and 7.

You need to have your PC connected to Internet as we will clone Caffe from the github repositories.

First save the **`install_caffe_scripts.tar`** in your home directory (represented by the tilde "~" symbol in Linux).

Now, execute the following Linux commands to install Caffe:

```
cd ~
tar –xvf install_caffe_scripts.tar
mkdir ~/caffe_tools
source ~/scripts/caffe/install_caffe.sh
```

Note that the script also solves two very recent issues I have found (documented in Section 9.3). If everything goes right, you should see something similar to the below screenshot of Figure 17:

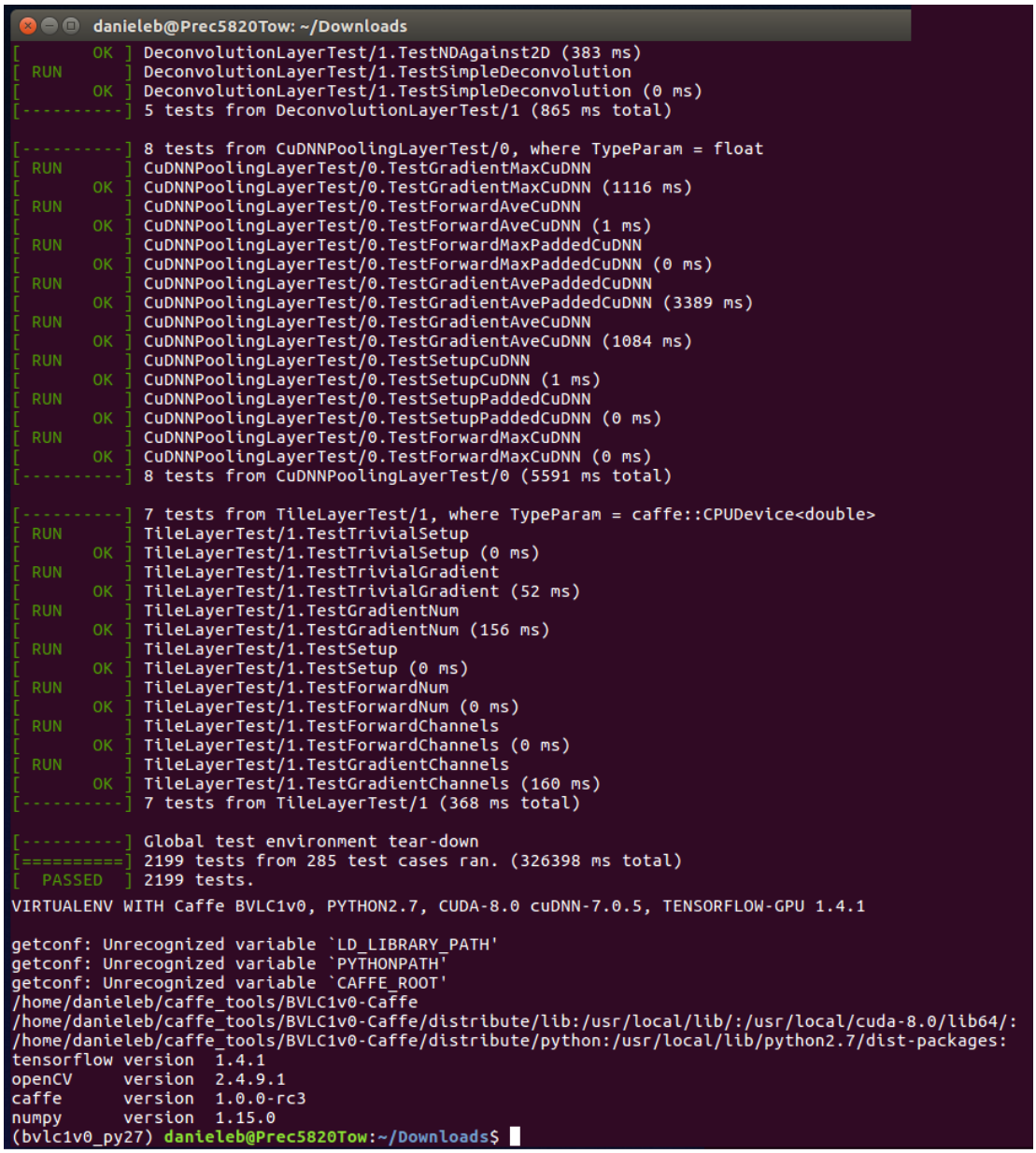

Figure 17: final fragment of the Caffe installation process

11.0 Install DeePhi' tools on the Host PC and Target board

DeePhi' tools for ML implementation on Xilinx FPGA -called DNNDK- are split in two parts, one running on the Ubuntu 16.04 PC (the Host) and another running on the ZCU102 board (the Target). The user has to build the image file to boot the Target ZCU102 board, which has to be connected via WLAN to the Host PC (see Section 5.3). Be sure the jumpers of ZCU102 are set as illustrated in Figure 18 and Figure 19 (see also https://www.xilinx.com/support/answers/69164.html).

The Host PC side of the toolchain has to be compatible with:

>> ubuntu 16.04 + CUDA 8.0 + cuDNN v7.0.5
>> ubuntu 16.04 + CUDA 9.0 + cuDNN v7.0.5
>> ubuntu 16.04 + CUDA 9.1 + cuDNN v7.0.5
>> libnccl.so.1 (follow the instructions from Section 7.3)

Download DeePhi tools and ZCU102 image files from the DeePhi website. In particular:

- **deephi_dnndk_v2.08_beta.tar.gz** from http://deephi.com/assets/deephi_dnndk_v2.08_beta.tar.gz
- **2018-12-04-ZCU102-desktop-stretch.img.zip** from http://www.deephi.com/assets/2018-12-04-zcu102-desktop-stretch.img.zip
- **DNNDK User Guide 1327** available from http://www.deephi.com/assets/ug1327-xilinx-dnndk-user-guide.pdf

Now you have to execute the following steps.

- On your Host PC open the **deephi_dnndk_v2.08_beta.tar.gz** file (assuming you have put it in `~/DNNDK`) with the following commands and prepare the files to be sent to the Target board later

```
1. cd ~/DNNDK
2. gzip -v -d deephi_dnndk_v2.08_beta.tar.gz
3. tar -xvf deephi_dnndk_v2.08_beta.tar
4. cd deephi_dnndk_v2.08_beta
5. tar -cvf common.tar ./common
6. tar -cvf ZCU102.tar ./ZCU102
7. gzip ZCU102.tar
```

  `ZCU102.tar.gz` and **`common.tar`** are the files you need to transfer from the Host PC to the Target board. But before doing it you have to prepare the SD card.

- Unzip **2018-12-04-ZCU102-desktop-stretch.img.zip** to get the image file **2018-12-04-ZCU102-desktop-stretch.img**. Insert an empty SD-Card (with size >= 16 GB) into your Host Ubuntu Linux PC

- You have now to flash the OS image on the SD card. You can use two alternative ways:

  1. Use the **Etcher** utility from a Windows OS PC, as described in pages 20-21-22 of Xilinx/DeePhi UG1327**.**

  2. Otherwise, use the following commands from an Ubuntu OS PC:

- **`sudo lsblk`**
  on the Linux PC and search for your ZCU102 SD-Card (in some Ubuntu PCs the sd-card is seen as **`/dev/ssd`** in others as **`/dev/mmcblk0`**); let us assume it is called **`/dev/sdd`**
- Burn the img file on the SD-Card with the Linux command:

  `sudo dd if=deephi_zcu102_dnndk_1.08.img of=/dev/ssd`

  (in case of troubles look at **https://learn.sparkfun.com/tutorials/sd-cards-and-writing-images).** Normally it takes about 10-30 min depending on your PC

- Once the above process has finished, you can boot the Target board.
- Open a puTTY terminal (if requested, login/password: root/root) with the usual Zynq parameters (115200,8,n,1,N) for the UART connection. Note that you have to be superuser, therefore call the PuTTY GUI with **`sudo`** or call it via the following command line:

  `sudo putty /dev/ttyUSB0 –serial –sercfg 115200,8,n,1,N`

- Connect a WLAN cable from Target board to the Host PC, then run the following commands:
  - (from Target) **`ifconfig eth0 192.168.1.100 netmask 255.255.255.0`**
  - (from Host) **`sudo ifconfig eth1 192.168.1.101 netmask 255.255.255.0`**
  - (from Host)
    - **`cd ~/DNNDK/deephi_dnndk_v2.08_beta`**
    - **`scp common.tar root@192.168.1.100:/root`**
    - **`scp ZCU102.tar.gz root@192.168.1.100:/root`**
  - (from Host)
    - **`cd ~/DNNDK/deephi_dnndk_v2.08_beta`**
    - **`cd host_x86`**
    - **`sudo ./install.sh ZCU102`**
  - (from Target)
    - **`cd /root`**
    - **`gzip –v –d *.gz`**
    - **`tar –xvf common.tar`**
    - **`tar –xvf ZCU102.tar`**
    - **`cd ZCU102`**
    - **`./install.sh`**

Once finished you should see something similar to what shown in Figure 20 and Figure 21 (which are related to older 2.0.7 release).

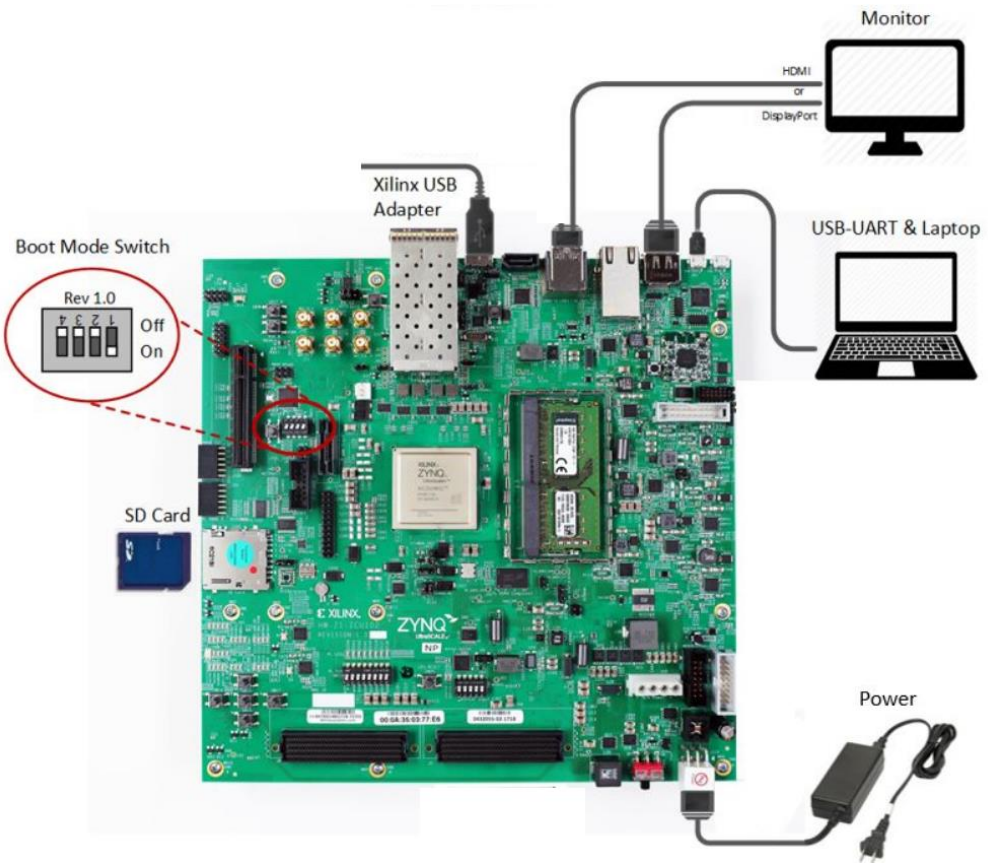

Figure 18: ZCU102 board setup for SD-Card booting

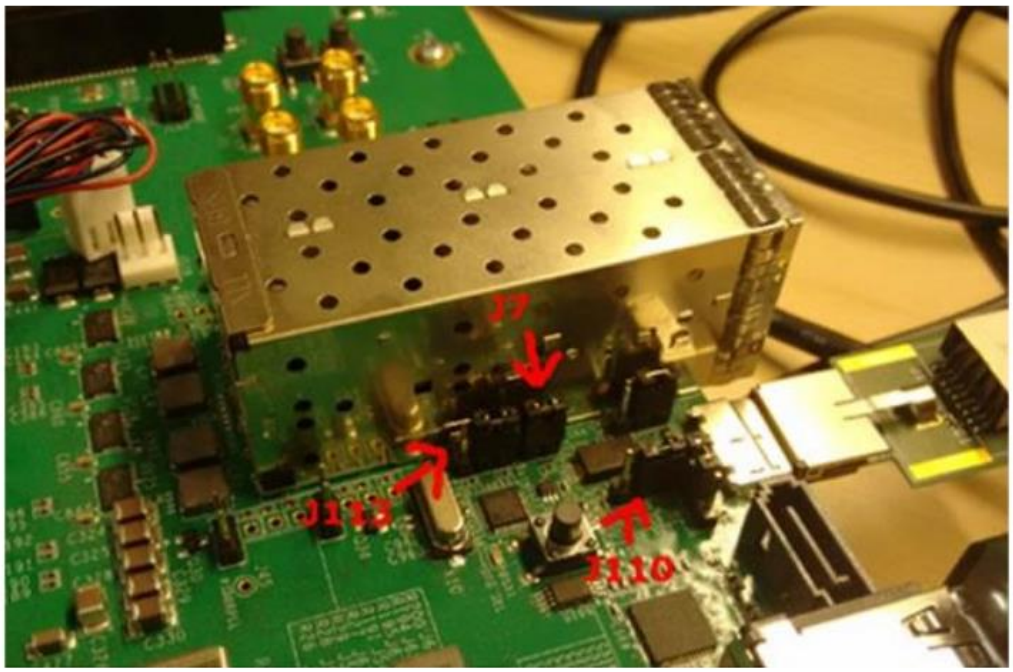

Figure 19: Correct setting of the jumpers

```
danieleb@Prec5820Tow: ~/ML/DNNDK/deephi_dnndk_v2.07_beta/host_x86
decent_libs_dependency.txt  install.sh  pkgs
danieleb@Prec5820Tow:~/ML/DNNDK/deephi_dnndk_v2.07_beta/host_x86$ ./install.sh ZCU102
Inspect system enviroment...
[system version]
No LSB modules are available.
Description: Ubuntu 16.04.5 LTS
16.04
[CUDA version]
8.0.61 Version
[CUDNN version]
7.0.5
Begin to install DeePhi DNNDK tools on host ...
cp: cannot create regular file '/usr/local/bin/decent': Permission denied
cp: cannot create regular file '/usr/local/bin/dnnc-dpu1.3.0': Permission denied
cp: cannot create regular file '/usr/local/bin/dnnc-dpu1.3.1': Permission denied
Installation failed!
danieleb@Prec5820Tow:~/ML/DNNDK/deephi_dnndk_v2.07_beta/host_x86$ sudo ./install.sh ZCU102
[sudo] password for danieleb:
Inspect system enviroment...
[system version]
No LSB modules are available.
Description: Ubuntu 16.04.5 LTS
16.04
[CUDA version]
8.0.61 Version
[CUDNN version]
7.0.5
Begin to install DeePhi DNNDK tools on host ...
Complete installation successfully.
danieleb@Prec5820Tow:~/ML/DNNDK/deephi_dnndk_v2.07_beta/host_x86$
```

Figure 20: DeePhi DNNDK 2.0.7 install on Host PC

```
/dev/ttyUSB0 - PuTTY
root@xlnx:~# cd ~
root@xlnx:~# ls
Desktop  ZCU102  common
root@xlnx:~# pwd
/root
root@xlnx:~# cd ZCU102/
root@xlnx:~/ZCU102# ls
install.sh  pkgs  samples
root@xlnx:~/ZCU102# ./install.sh
Begin to install DeePhi DNNDK ...
Install DeePhi DPU Driver ...
[  104.430510] dpu: loading out-of-tree module taints kernel.
[  104.436823] [DPU][3268]Checking DPU signature at addr = 0x8ff00000,
[  104.443202] [DPU][3268]DPU signature checking done!
Install DeePhi tools, runtime & libraries ...
Complete installation successfully.
root@xlnx:~/ZCU102# dexplorer -v
DNNDK version  2.07 beta
Copyright @ 2016-2018 DeePhi Inc. All Rights Reserved.

DExplorer version 1.5
Build Label: Oct 12 2018 12:00:01

DSight version 1.4
Build Label: Oct 12 2018 12:00:02

N2Cube Core library version 2.1
Build Label: Oct 12 2018 12:00:22

DPU Driver version 2.0
Build Label: Oct 12 2018 11:59:58
root@xlnx:~/ZCU102#
```

Figure 21: DeePhi DNNDK 2.0.7 install on Target board

12 Setting a p2.xlarge EC2 instance on an AWS Ubuntu 16.04 AMI

12.1 Prepare the DeePhi DNNDK Host tools for the AWS

Assuming you have correctly run Section 11, you can prepare the DeePhi DNNDK Host tools archive to copy them later on the AWS. To this purpose just execute the following commands:

```
cd ~/DNNDK/deephi_dnndk_v2.08_beta
cd host_x86/pkgs/ubuntu16.04
mkdir tools
cp dnnc-dpu* ./tools
cp cuda_8.0.61_GA2_cudnn_v7.0.5/decent ./tools
tar -cvf deephi_host_208tools.tar ./tools
gzip deephi_host_208tools.tar
mv deephi_host_208tools.tar.gz $HOME/ML/DNNKK
```

To further check the install was correct on your Host PC side, you can launch the following commands:

```
cd $HOME/ML/DNNDK/
tar -xvf deephi_host_208tools.tar.gz
cd tools
ln -s dnnc-dpu1.3.0 dnnc
decent --version
dnnc --version
```

and you should get the screenshot of Figure 22, related to the 2.08 release.

```
(bvlc1v0_py27) danieleb@Prec5820Tow:~/ML/DNNDK/tools$ ls -l
total 65520
-rwxr-x--- 1 danieleb danieleb 15197856 gen  8 20:53 decent
lrwxrwxrwx 1 danieleb danieleb       13 gen  8 21:01 dnnc -> dnnc-dpu1.3.0
-rwxr-x--- 1 danieleb danieleb 25500904 gen  8 20:52 dnnc-dpu1.3.0
-rwxr-x--- 1 danieleb danieleb 26389736 gen  8 20:52 dnnc-dpu1.3.7
(bvlc1v0_py27) danieleb@Prec5820Tow:~/ML/DNNDK/tools$ decent --version
decent version 1.2.4
Build Label Dec  7 2018 02:47:58
(c) Copyright 2016 - 2018 Xilinx, Inc. All rights reserved.
(bvlc1v0_py27) danieleb@Prec5820Tow:~/ML/DNNDK/tools$ dnnc --version
dnnc version v2.03
DPU Target : v1.3.0
Build Label: Dec 11 2018 11:52:13
Copyright @2018 Xilinx Inc. All Rights Reserved.
(bvlc1v0_py27) danieleb@Prec5820Tow:~/ML/DNNDK/tools$
```

Figure 22: check the DeePhi 2.08 Host tools

12.2 Take the p2.xlarge EC2 Instance on Ubuntu 16.04 AMI of AWS

I am assuming you are already familiar with AWS, if not I strongly recommend you to have a look at this tutorial (53min): https://www.youtube.com/watch?v=1Eh1uxLyXJ8.

The movie is a little old but it is still good if you start from zero, as it gives you info about:

- Setup AWS account
- Create IAM user and enable MFA
- Create SSH key pairs
- Start an EC2 instance
- S3: Simple Storage System
- EBS: Elastic Block Storage (do snapshots to never lose content when you stop EC2)
- AMI: Amazon Machine Images (bootable examples of snapshots).

I cannot give you more help here about Getting Started with AWS, as this would be out of the scope of this tutorial. I also recommend you to create an S3 bucket to store your files permanently, out of any EC2 instance (have in mind that moving large files from your local PC to the EC2 instance is generally slower than moving them from your local PC to the S3 bucket and then from the S3 bucket to your EC2 instance).

You have to select the **Deep Learning AMI** illustrated in Figure 23 and then you need to take a **p2.xlarge EC2** instance shown in Figure 24, being it equipped with a NVIDIA K80 GPU**.**

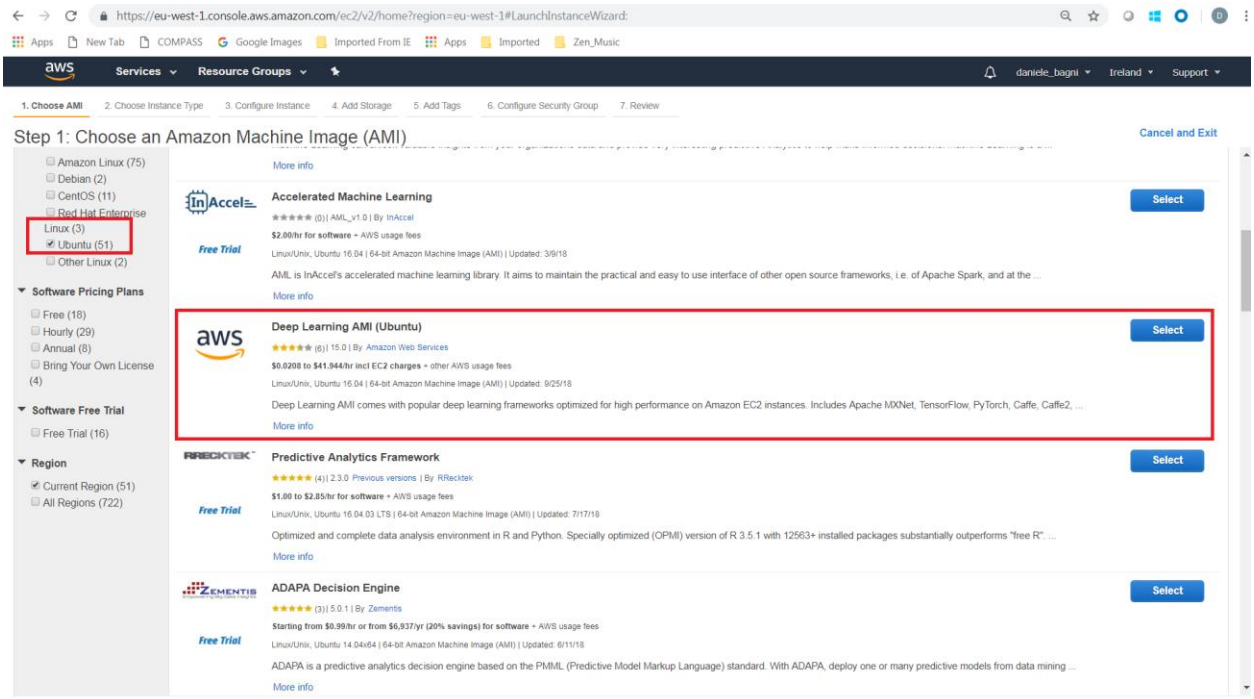

Figure 23: Deep Learning AMI

Once you log in the AWS, you will see the screenshot illustrated in Figure 25. Note that you will need to use **caffe_p27** and **tensorflow_p27** virtual environments to run my **CIFAR10-ML-Tutorial**, respectively with the one of the commands

- `source activate caffe_p27`
- `source activate tensorflow_p27`

If you execute the following Linux command you can see what there is in the Caffe Python2.7 virtual environment, as illustrated in Figure 26:

- `cd`
- `ls src/caffe_python_2`

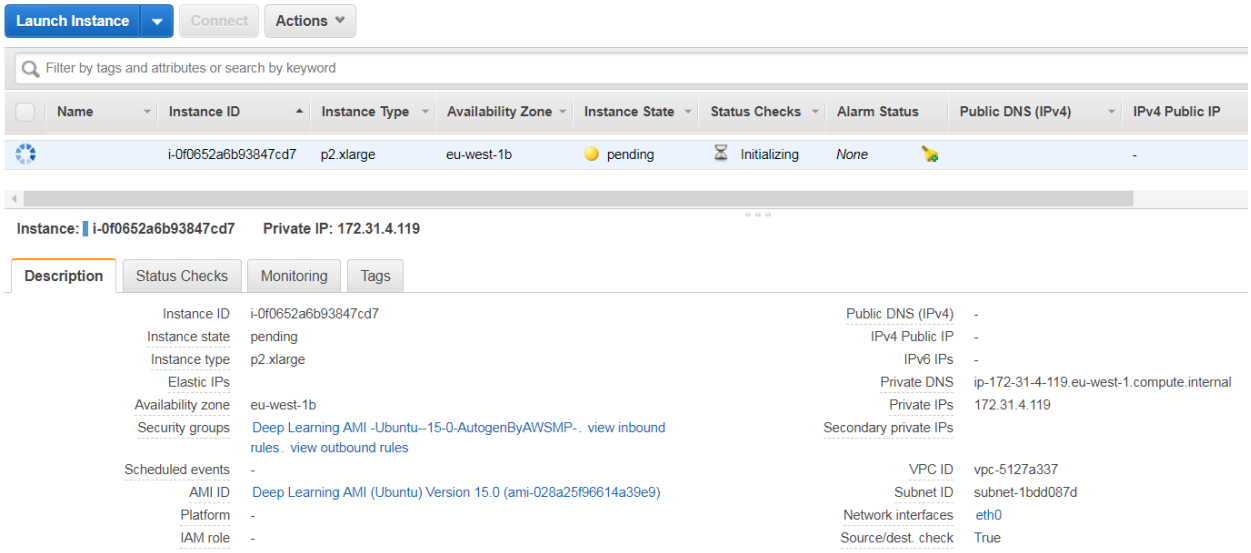

Figure 24: p2.xlarge EC2 instance

Note that Caffe on AWS has a slightly different installation structure, being based on **anaconda** (www.anaconda.com). In particular:

- folder **`distribute`** does not exists and is replaced by folder **`build/install`**
- **`caffe.bin`** executable is just named **`caffe`**
- **`compute_image_mean.bin`** has not been compiled with OpenCV: you should recompile the whole Caffe on AWS, do not do that, too much time consuming and error prone process!

Furthermore, from **tensorflow_p27** environment (**`source activate tensorflow_p27`**) I also had to run the following commands (the first one only once forever)

- **`conda install keras`**
- **`export LD_LIBRARY_PATH=/home/ec2-user/src/caffe_python_2/build/install/lib64/:$LD_LIBRARY_PATH`**
- **`export PYTHONPATH=/usr/local/lib/python2.7/dist-packages/:$PYTHONPATH`**

All those settings have been captured in the script set_aws_ML_env.sh. Besides that, the script also creates a system of directories named:

1) **`/caffe`**
2) **`/home/ML`**
3) **`/home/ML/DNNDK/tools`**

in order to prepare the AWS environment for my ML tutorials (CIFAR10 and CATSvsDOGs). To avoid wasting storage space I am doing heavy usage of soft links.

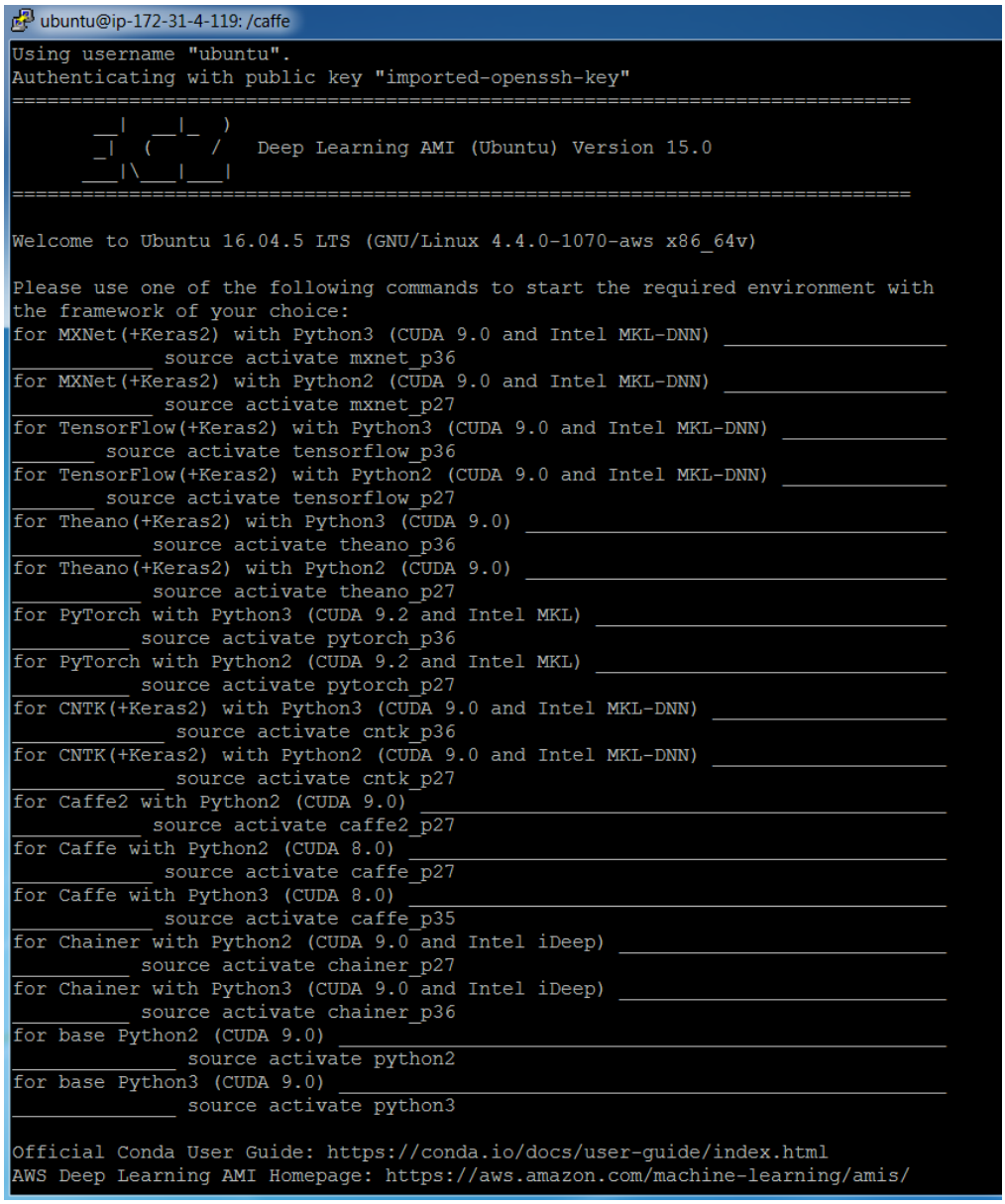

Figure 25: initial page of AWS

```
[ec2-user@ip-172-31-10-57 ~]$ ls -l src/caffe_python_2/
total 124
drwxrwxr-x 14 ec2-user ec2-user  4096 19 lug 20.02 build
-rw-rw-r--  1 ec2-user ec2-user  1180 19 lug 20.01 caffe.cloc
drwxrwxr-x  5 ec2-user ec2-user  4096 19 lug 20.01 cmake
-rw-rw-r--  1 ec2-user ec2-user  4197 19 lug 20.01 CMakeLists.txt
-rw-rw-r--  1 ec2-user ec2-user  1917 19 lug 20.01 CONTRIBUTING.md
-rw-rw-r--  1 ec2-user ec2-user   620 19 lug 20.01 CONTRIBUTORS.md
drwxrwxr-x  5 ec2-user ec2-user  4096 19 lug 20.01 data
drwxrwxr-x  4 ec2-user ec2-user  4096 19 lug 20.01 docker
drwxrwxr-x  6 ec2-user ec2-user  4096 19 lug 20.01 docs
drwxrwxr-x 15 ec2-user ec2-user  4096 19 lug 20.01 examples
drwxrwxr-x  3 ec2-user ec2-user  4096 19 lug 20.01 include
-rw-rw-r--  1 ec2-user ec2-user   210 19 lug 20.01 INSTALL.md
-rw-rw-r--  1 ec2-user ec2-user  2092 19 lug 20.01 LICENSE
-rw-rw-r--  1 ec2-user ec2-user 24041 19 lug 20.01 Makefile
-rw-rw-r--  1 ec2-user ec2-user  4634 19 lug 20.01 Makefile.config
-rw-rw-r--  1 ec2-user ec2-user  4631 19 lug 20.01 Makefile.config.example
drwxrwxr-x  5 ec2-user ec2-user  4096 19 lug 20.01 matlab
drwxrwxr-x  7 ec2-user ec2-user  4096 19 lug 20.01 models
drwxrwxr-x  3 ec2-user ec2-user  4096 19 lug 20.01 python
-rw-rw-r--  1 ec2-user ec2-user  2130 19 lug 20.01 README.md
drwxrwxr-x  3 ec2-user ec2-user  4096 19 lug 20.01 scripts
drwxrwxr-x  4 ec2-user ec2-user  4096 19 lug 20.01 src
drwxrwxr-x  3 ec2-user ec2-user  4096 19 lug 20.01 tools
```

Figure 26: Python2.7-based Caffe on AWS

12.3 Work on your AWS p2.xlarge EC2

First thing to do is to copy the file **deephi_host_208tools.tar** from your local PC to your AWS S3 bucket as illustrated for my case in Figure 27. Once done that you can copy them on your $HOME in the AWS as illustrated in Figure 28.

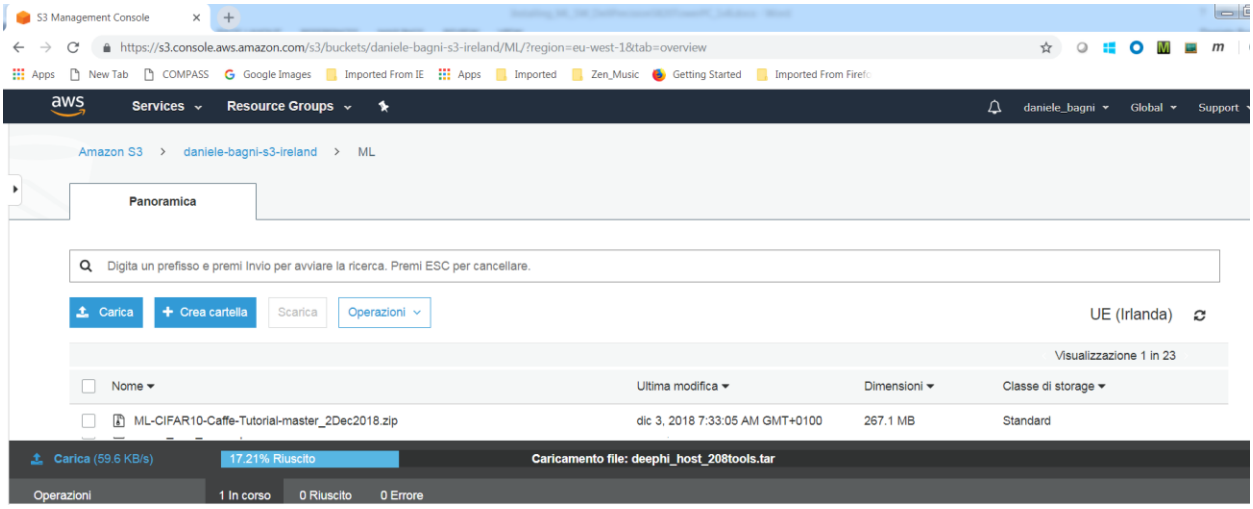

Figure 27: Copy the DNNDK 2.0.8 Host tools from your local PC to your AWS S3 bucket

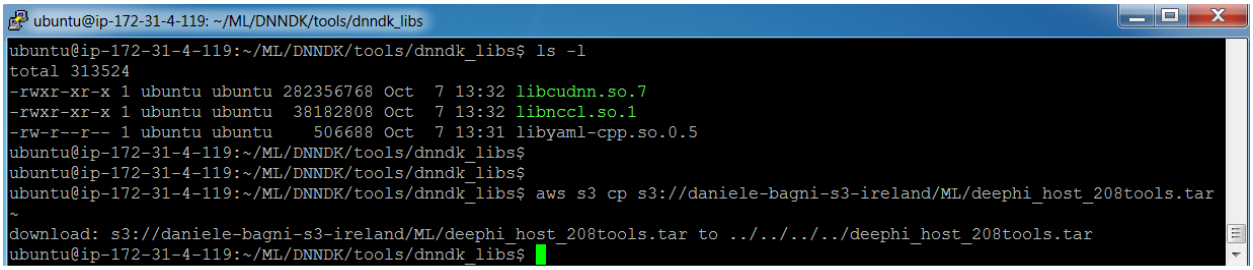

Figure 28: copying from AWS S3 to AWS EC2 home

Note that Figure 28 also shows three libraries needed by DNNDK and missing on the AWS, they are:

`libcudnn.so.7, libnccl.so.1, libyaml-cpp.so.0.5`

All together they take more than 282 MB size. I have compressed them with gzip and then archived them in a single tar file which then I have split in chunks of less than 25 MB size (the maximum size limit for a file in GitHub) and placed in the folder **dnndk_host** of this repository and uploaded them individually into my S3 bucket and from there to my AWS $HOME.

Copy all those files to your AWS $HOME together with the 2 scripts **aws_activate_dnndk.sh** and **set_aws_M_env.sh** placed in the folder **aws_scripts**. Once copied all these files on your AWS you should have a situation like the one depicted in the top part of Figure 29.

Now launch the following commands from your AWS $HOME:

- **`cd ~`**
- **`source ./set_aws_ML_env.sh`**

At the end of the process you should see what illustrated in the bottom part of Figure 29. For your reference, I have saved what I got on my terminal in a log file named **aws_logfile.txt**, placed in the folder **aws_scripts.**

Every time you have to launch the DNNDK tools on the AWS use the commands

- **`cd ~`**
- **`source ./aws/activate`**

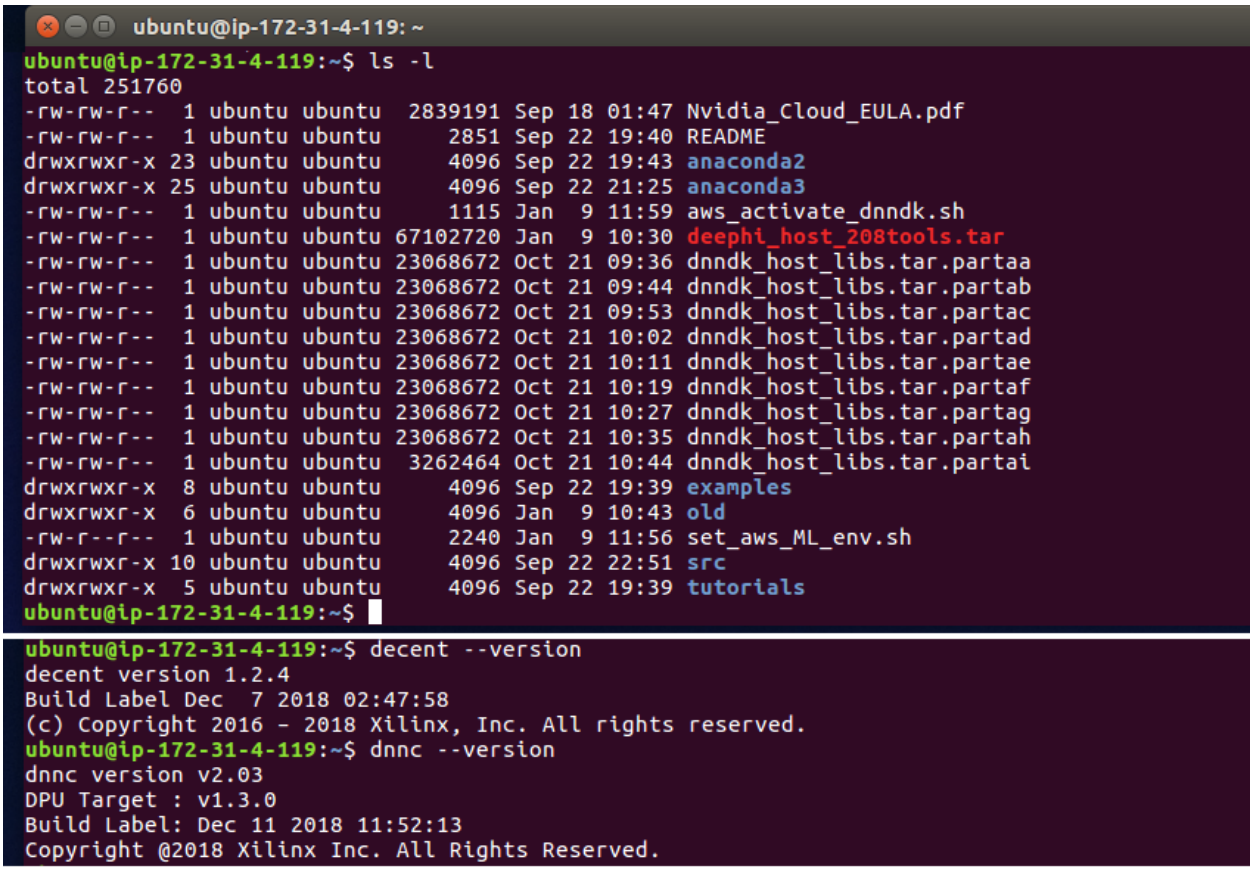

Figure 29: starting point of DNNDK install on the AWS (top) and final verification (bottom)