# Research Jetpack SDK

04 November 2020 11:06

**© RAJAT KUMAR**
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Objective : We need to find out whether our current Raspberry Pi demo would be CUDA accelerated for Bbth face and mask detection. Jetpack SDK supports both opencv and tensor. We need to find out if our code needs to be modified to use jetpack and what are the modifications. For example, opencv and tensor support may have differences that require code changes from our side.

GPU Accelerated Computing with Python:-
Numba, a Python compiler from Anaconda that can compile Python code for execution on CUDA-capable GPUs, provides Python developers with an easy entry into GPU-accelerated computing and a path for using increasingly sophisticated CUDA code with a minimum of new syntax and jargon. With CUDA Python and Numba, you get the best of both worlds: rapid iterative development with Python combined with the speed of a compiled language targeting both CPUs and NVIDIA GPUs.

SETUP CUDA PYTHON:
To run CUDA Python, you will need the CUDA Toolkit installed on a system with CUDA capable GTo get started with Numba, the first step is to download and install the Anaconda python distribution that includes many popular packages (Numpy, Scipy, Matplotlib, iPython, etc) and "conda", a powerful package manager. Once you have Anaconda installed, install the required CUDA packages by typing conda install numba cudatoolkit pyculib. PUs.

You can't run all of your python code in GPU. You have to write some parallel python code to run in CUDA GPU or use libraries which support CUDA GPU.
Your best bet for rewriting custom code is Numba. GPU Accelerated Computing with Python
If it is for deep learning, use tensorflow, or pytorch or keras. Make sure to follow install instructions for CUDA GPU. (I would suggest to use nvidia docker images).
If it is for machine learning and ETL, use RAPIDS.

"""
CPU/GPU Data Transfer
To transfer data from GpuMat to Mat and vice-versa, OpenCV provides two functions:

upload, which copies data from host memory to device memory
download, which copies data from device memory to host memory.

Utilizing Multiple GPUs
By default, each of the OpenCV CUDA algorithms uses a single GPU. If you need to utilize multiple GPUs, you have to manually distribute the work between GPUs. To switch active device use cv::cuda::setDevice (cv2.cuda.SetDevice) function.
"""

Getting Started with OpenCV CUDA Module: Checkout this:
https://www.learnopencv.com/getting-started-opencv-cuda-modul/

Tensorflow Changes:

```
sudo apt-get update
sudo apt-get install python-pip python-dev
pip install tensorflow-gpu
```
install the Cuda Toolkit.

Steps:

Uninstall your old tensorflow
Install tensorflow-gpu using pip install tensorflow-gpu
Install Nvidia Graphics Card & Drivers (you probably already have)
Download & Install CUDA
Download & Install cuDNN
Verify by simple program
```
from tensorflow.python.client import device_lib
 print(device_lib.list_local_devices())
```

To Check GPU Availability in Tensorflow
```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    print("Name:", gpu.name, "  Type:", gpu.device_type)
```

The changes to your Tensorflow code should be minimal. If a TensorFlow operation has both CPU and GPU implementations, the GPU devices will be prioritized when the operation is assigned to a device.
If you would like a particular operation to run on a device of your choice instead of using the defaults, you can use with tf.device to create a device context. This forces all the operations within that context to have the same device assignment.

Examples:
```
with tf.device('/gpu:0'):
 a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
 b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
with tf.device('/gpu:0'):
    a = tf.placeholder(tf.float32, [10000, 10000])
    b = tf.placeholder(tf.float32, [10000, 10000])
    # Compute A^n and B^n and store results in c1
    c1.append(matpow(a, n))
    c1.append(matpow(b, n))
with tf.device('/cpu:0'):
  sum = tf.add_n(c1) #Addition of all elements in c1, i.e. A^n + B^n
```

References :

Hi,
Please find attached changed code for running tensorflow on gpu.

Steps:

Uninstall your old tensorflow
Install tensorflow-gpu using pip install tensorflow-gpu

As of the 20.02 TensorFlow release, the package name has changed from tensorflow-gpu to tensorflow

Install Nvidia Graphics Card & Drivers (you probably already have)
Download & Install CUDA
Download & Install cuDNN
Verify by simple program

For installation follow this:
https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html

To Check GPU Availability in Tensorflow
```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    print("Name:", gpu.name, "  Type:", gpu.device_type)
```

https://github.com/bafu/cv2cuda
https://developer.nvidia.com/embedded/jetpack
https://developer.nvidia.com/how-to-cuda-python?ncid=afm-chs-44270&ranMID=44270&ranEAID=a1LgFw09t88&ranSiteID=a1LgFw09t88-2IgCmWJJvc1VnxpVxBd7LQ
https://www.quora.com/How-can-I-use-my-Python-code-with-CUDA-GPU-instead-of-CPU
https://www.learnopencv.com/getting-started-opencv-cuda-modul/
https://weeraman.com/put-that-gpu-to-good-use-with-python-e5a437168c01
https://towardsdatascience.com/how-to-traine-tensorflow-models-79426dabd304
https://stackoverflow.com/questions/51306862/how-do-i-use-tensorflow-gpu