gyrfalcon technology

GTI 2801 / GTI 2803 / GTI 5801

Neural Network Accelerator

Software Development Kit (SDK)

Version 4.5.1.0

Contents

1. About this Document

This document contains information that is proprietary and confidential to Gyrfalcon Technology Inc. and is intended for the specific use of the recipient, for the purpose of evaluating or using Gyrfalcon Technology's products and/or IP. This document is provided to the recipient with the expressed understanding that the recipient will not divulge its contents to other parties or otherwise misappropriate the information contained herein.

Contact Gyrfalcon
Silicon Valley HQ
Gyrfalcon Technology, Inc.
1900 McCarthy Boulevard, Suite 208
Milpitas, CA 95035 USA
www.gyrfalcontech.ai

2. Overview

Gyrfalcon Technology Inc.'s (GTI) Artificial Intelligence (AI) network Software Development Kit (SDK) package includes hardware libraries that drives Gyrfalcon's proprietary silicon solutions and support libraries for building customizable AI networks and inference applications. It provides efficient hardware acceleration using GTI's low-power GTI 2801, GTI 2803 and GTI5801 processors to perform functions used in video/image recognition, voice recognition, scene recognition, and autonomous vehicle systems (cars, trucks, drones, and more).

This document contains the following information:

3. Release Notes

3.1 Special Notes

- This release still supports legacy (SDK v3.1) APIs and the way it used to run.

3.2 Major Changes

For GTI 5801:

- Changed default USB chunk size 64k read/write delay to 20us.

For ARMv7 platform:

- Fixed an issue that running the reference applications results in an alignment trap/exception on ARMv7 platforms.

For GTI 2803:

- Changed default PLL clock rate from 250MHz to 200MHz, to avoid stability issues on GTI 2803 devices with PCIe interface.
    - Updated GTI 2803 register definitions.

Application changes

For Apps/Pipeline demo:

- Fixed a failure on locking the second device.
- Corrected the pipeline demo performance (FPS) measurements.
- Added performance measurement -p option, that prints the average performance results over multiple inferences.

For Apps/liteDemo:

- Enabled liteDemo to output raw results that are used by Jenkins.

For all applications added chrono library support for better precision of timestamp calculations.

For PCIe driver support:

- For PCIe DMA driver - use semaphore to speed up DMA read/write operations.
- Fixed a typo on the FPGA device ID in /Drivers/Linux/pcie_drv/gti_drv.c file.

- The PCIe DMA driver to use mapped memory for the read operation.

3.3 License

- See the GTI SDK license and third-party license disclaimers in the root directory.
    - Third-party terms and open source terms.
    - Terms of service.
    - End user license agreement.

4. GTI SDK Support for Linux OS

4.1 Overview

Gyrfalcon Technology Inc provides hardware and software for application development targeted to AI and machine learning. The software development kit (SDK) includes drivers, libraries and source code for AI learning and inferencing. GTI solutions guarantee low cost, low power and high performance.

The Linux package includes:

- APIs and libraries to support GTI 2801, GTI 2803 and GTI 5801.
- Device driver/library for FTDI/PCIe interface.
- Reference application (written in C++ and Python) for video/image recognition.
- Data package (includes sample data and model files).

Equipment to be supplied by user:

- Linux (Ubuntu 16.04, 64-bit version).
- Intel i5 or better processor (i7 preferred) 3.0 GHz or faster.
- 8 GB or more RAM.
- At least one USB 3.0 port.

Third party software:

- Python 2.7
- OpenCV 3.2.0
- GNU compiler (minimum required g++ version 4.9, preferred 5.4).

4.2 SDK File Structure

The GTI SDK package is delivered in a tarball format that you decompress to your local file system. Following are the unpacking instructions for Ubuntu 16.04 (Linux):

```
tar zxvf GTISDK-Linux_x86_64-4.x.x.x.tgz
```

The resulting folder structure is shown below.

Note: Depending on the release version, there may be slight variations in this folder structure.

GTISDK

```
.
├── Apps
│   ├── Android
│   │   ├── app
│   │   └── jni
│   ├── Data
│   │   ├── Image_bmp_c1000
│   │   ├── Image_bmp_c20
│   │   ├── Image_bmp_c40
│   │   ├── Image_dog
│   │   ├── Image_lite
│   │   └── Image_mp4
│   ├── Demo
│   ├── liteDemo
│   │   └── Android
│   ├── Models
│   │   ├── 2801
│   │   ├── 2803
│   │   └── 5801
│   ├── PipelineDemo
│   └── Python
├── Bin
│   └── Linux
│    └── x86_64
├── Build
├── data
│   ├── Image_Lite
│   └── Models
│    └── gti2801
├── Documents
├── Drivers
│   └── Linux
│    ├── pcie_driver_2801
│    └── pcie_drv
├── Firmware
│   └── 5801
│    └── Rev
├── Include
│   └── Android
│    └── jni
```

```
├──── Lib
│     ├──── Android
│     │     ├──── arm64-v8a
│     │     └──── armeabi-v7a
│     └──── Linux
│       ├──── OpenCV
│       └──── x86_64
├──── Python
│     ├──── Lib
│     │     ├──── gtiClassify
│     │     └──── gtilib
│     └──── Samples
├──── Samples
│     ├──── Android
│     │     ├──── app
│     │     └──── jni
│     └──── Sample
├──── scripts
├──── Thirdparty
│     ├──── download
│     ├──── libusb
│     ├──── libusb-aarch64
│     ├──── libusb-android
│     │     └──── patches
│     └──── libusb-armv7l
└──── Tools
      ├──── eusbTool
      ├──── hwTool
      │     └──── Android
      ├──── imageTool
      ├──── modelTool
      └──── usbTool
      └──── Android
```

- **Apps** : contains reference applications, data and models
  - **Apps/Android** : an image inference demo for Android platforms.
  - **Apps/Demo**: fully featured reference application for inferencing, you can modify and re-compile your own code **(x86_64 release package only)**
  - **Apps/liteDemo**: lite version of GTI SDK reference application supporting image inference only
  - **Apps/PipelineDemo**: a demo pipelining feature for image inferencing.
  - **Apps/Python**: reference application in Python **(x86_64 release package only)**

  - data/Models: models with a new format for SDK 4.0 and onwards, that's different from legacy models under **data/Model**.
  - **Apps/Models** : includes models to support applications based on GTI 2801 (**Apps/Models/2801**) and GTI 2803 (**Apps/Models/2803**) and GTI 5801 (**Apps/Models/5801**).
  - **Apps/Data** : includes images that supply the applications with sample input data.
- **Bin** : pre-built executables.
- **Build** : GTI SDK environment setup files
  - **Build/GTISDK.env** : GTI SDK build configuration file, now supports four different architectures (CPU_ARCH - x86_64, ARMv7l, AArch64)
- **data** : **This directory applies to the legacy mode only.** Contains input image and model for the legacy Samples applications.
- **Drivers** : provides support for communication interfaces, including PCIe and USB.
  - **Drivers/Linux/pcie_driver_2801** : GTI 2801 PCIe device driver for legacy support **(x86_64 release package only)**
  - **Drivers/Linux/pcie_drv** : PCIe device driver for GTI 2801 and GTI 2803 **(x86_64 release package only)**
- **Include** : header files.
- **Lib** : supporting libraries for communication interfaces and GTI 2801/GTI 2803 chip support.
  - **Lib/Android** : pre-built libraries for Android support.
  - **Lib/Linux/OpenCV/opencv2** : OpenCV header files (OpenCV Copyright, x86_64 only)
  - **Lib/Linux/OpenCV/x86_64** : pre-built libraries for GTI SDK **(OpenCV Copyright, x86_64 release package only)**
  - **Lib/Linux/x86_64** : pre-built libraries **(GTILibrary and FTDI, x86_64 release package only)**
  - **Lib/Linux/armv7l** : pre-built libraries **(GTILibrary and FTDI, armv7l release package only)**
  - **Lib/Linux/aarch64** : pre-built libraries **(GTILibrary and FTDI, aarch64 release package only)**
- **Python** : libraries and sample code in Python.
  - **Python/Lib** : libraries for Python application support.
  - **Python/Sample** : Reference applications in Python **(x86_64 release package only)**
- **scripts** : dependency installation scripts (for legacy SDK 3.1 support).
- **Samples/Sample**: **This directory applies to the legacy mode only.** GTI SDK reference application for legacy GTI SDK v3.1 support.

- **Tools** : host and target tools for image, model conversion, and checking host/target communication flow.

4.3 Installation

4.3.1 Default installation with SourceMe.env script

The SDK package comes with all needed data packages (if you are not using the Legacy model formats).

Prepare a Ubuntu 16.04 PC, extract the SDK release package and run the following command in a console terminal.

```
source SourceMe.env
```

The script SourceMe.env should automatically detect missing dependencies, such as pip, Python, interface drivers, and prompt you the necessary steps to take to complete the installation.

4.3.2 Manual installation

Note, if for any reason the SDK and library paths are not configured after running the environment setup file (`SourceMe.env`). Manually check on the following:

4.3.2.1 Setup path

- LD_LIBRARY_PATH, For example:

```
export LD_LIBRARY_PATH
=$PWD/Lib/Linux/OpenCV/x86_64/:$PWD/Lib/Linux/x86_64/:$LD_LIBRARY_PATH
```

- PYTHONPATH (if you are using Python), For example:

```
export PYTHONPATH=$PWD/Lib/Linux/OpenCV/x86_64/
```

4.3.2.2 System packages

On a Ubuntu 16.04 PC, use “apt” command to install the following packages and their dependencies:

```
libx11-dev
libgtk2.0-dev
pkg-config
libavcodec-dev
```

```
libavformat-dev
libswscale-dev
libjpeg-dev
python-numpy
python-tk
```

4.3.2.3 Applicable udev rules

Copy the appropriate rules files included in the SDK package to the system directory, for example, on Ubuntu 16.04, copy to:

```
/etc/udev/rules.d/
```

The following rules files are available under `Lib/Linux/<CPU_TYPE>/`

```
51-ftd3xx.rules  - for  FTDI USB devices
52-gtiusb.rules - for Direct USB devices
70-gti.rules - for PCIe devices
```

You don't need any device driver for FTDI, or Direct USB if running on a platform that supports USB, such as Ubuntu 16.04 Linux PC (see section 4.3.3 Installing Devices, for more details).

4.3.2.4 PCIe driver

Install PCIe driver, only if needed, if using PCIe card, such as the GAINboard. Run the following command to install:

```
cd Drivers/${OS_TYPE}/pcie_drv/
sudo make install
sudo modprobe gti_pcie_drv   #or reboot your PC
```

See section 4.3.3 Installing Devices, for more details.

4.3.2.5 PIP packages (Python only)

GTILibrary has a Python binding named "gtilib". Install it with the following command, if you're using Python applications:

```
python -m pip install --user Python/Lib/gtilib/
```

4.3.3 Installing Devices

The system overview of the GTI SDK along with supported hardware is provided in Figure 1.

GTI SDK is supported on various operating systems and reference applications are written in C/C++, Python and Java (for Android).

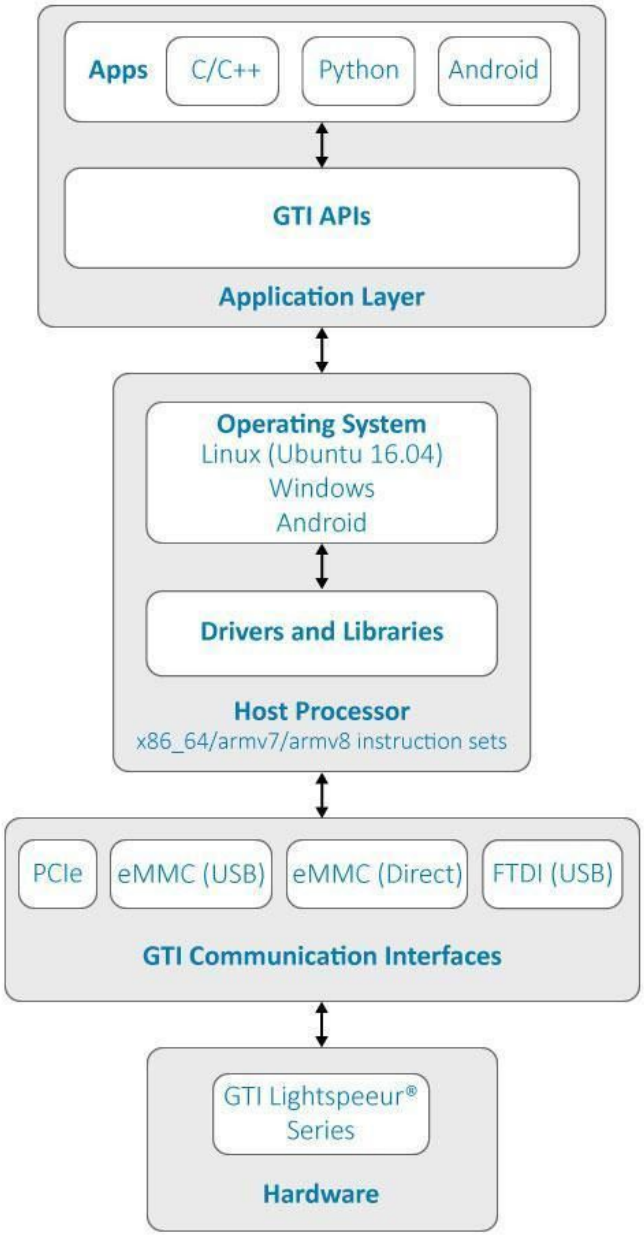

Figure 1. System overview

The GTI 2801/2803 chip supports FTDI/USB/PCIe interface. It's required to install a device driver before running the reference application for PCIe interface on each target platform. The

following is the list of supported architectures and host/target communication interfaces for each chip.

1) Supported platforms and interfaces for GTI 2801.

	Linux x86-64	**ARM-v7**	**ARM-v8**	**MIPS**	**Android-v7**	**Android-v8**	**Windows 7/10 x86-64**
FTDI USB	Yes	Limited support	Limited support	No	Yes	Yes	Yes
PCIe	Yes	Limited support	Limited support	No	No	No	No

2) Supported platforms and interfaces for GTI 2803.

	Linux x86-64	**ARM-v7**	**ARM-v8**	**MIPS**	**Android-v7**	**Android-v8**	**Windows 7/10 x86-64**
FTDI USB	Yes	Limited support	Limited support	No	Yes	Yes	Yes
PCIe	Yes	Limited support	Limited support	No	No	No	No

3) Supported platforms and interfaces for GTI 5801.

	Linux x86-64	**ARM-v7**	**ARM-v8**	**MIPS**	**Android-v7**	**Android-v8**
DirectUSB	Yes	Yes	Yes	No	Yes	Yes

PCIe	No	No	No	No	No	No

Notes:

Yes: fully supported.

No: not supported.

Limited support: supported and tested, but requires GTI and customer collaboration.

Install FTDI driver

Install FTDI UDEV configuration on x86_64 or armv7 target.

For **x86_64** Ubuntu system, run the following shell command from the target terminal.

```
sudo cp -i GTISDK/Lib/Linux/x86_64/51-ftd3xx.rules /etc/udev/rules.d/
sudo udevadm control --reload
```

On **armv7** target, run the following shell command from the target terminal.

```
sudo cp -i GTISDK/Lib/Linux/armv7l/51-ftd3xx.rules /etc/udev/rules.d/
sudo udevadm control --reload
```

Install PCIe driver

This is for Ubuntu 16.04 (x86_64) use only. Please contact GTI customer support, if you are using armv7, or aarch64 target.

SDK 4.0 and later releases use "`pcie_drv`" driver to support both GTI 2801 and GTI 2803 chips; legacy PCIe driver support is at "`pcie_driver_2801`" GTI 2801 support only.

When running SDK v4.0 and later, plug in GTI PCIe card and run the following shell command from terminal:

```
cd GTISDK/Drivers/Linux/pcie_drv/
sudo make install
sudo modprobe gti_pcie_drv
```

Instead of `modprobe` you can also reboot the computer to activate the driver.

To check device availability once drivers are installed, run "lsusb" for FTDI and run "lspci" for PCIe interfaces.

For more information on legacy SDK v3.1 driver installation, see section "8. Appendix: Support for Legacy SDK 3.1. " FTDI driver installation is the same.

4.4 Running Prebuilt Applications on GTI 2801 / GTI 2803 (On Linux Platforms)

The GTI SDK provides a demonstration application folder; **Apps**.

- **/Apps/Android/** : Reference application for image classification on Android device.
- **/Apps/Demo/demo** : Reference application with a new model format for image, video and camera classifications.
- **/Apps/litedemo/litedemo** : Reference application with a new model format for image classification.
- **/Apps/PipelineDemo/**: Reference application of Pipeline processing based on `GtiEvaluateWithCallback()` API.

Before running the prebuilt applications, GTI device must be plugged in. Supported hardware includes: USB device, GAINBOARD 2801, and GAINBOARD 2803. PCI cards typically require professional installation. The USB device can be plugged into a USB 3.0 port.

4.4.1 Enhancing the Application Performance

GTI SDK offers a configurable environment variable, OMP_NUM_THREADS, to enable parallel programming on your specific platform. This variable is added for openMP support (https://www.openmp.org/) that targets application optimization for better performance.

To determine the correct setting for OMP_NUM_THREADS, configure it so that the total number of active threads does not exceed the maximum number of threads of your computer. For example, to test mobilenet with 16 chips running simultaneously, each chip carrying one model for inference, in GTI tests, we launch 16 processes with each utilizing OMP_NUM_THREADS threads, therefore, the result of 16*OMP_NUM_THREADS should be less than or equal to the maximum number of threads of the system.

An example setting to run Apps/liteDemo application with Mobilenet model on Linux86_64 PC with 32 cores is shown below:

Number of GTI chips	OMP_NUM_THREADS
16	1
8	3
4	7
1	16

4.4.2 Configuring the FTDI dongle for GTI 2801 and GTI 2803

The device node of the GTI FTDI USB dongles are identified and configured automatically. Plug in the dongle and check for new devices with command **`lsusb`**, if unsure.

4.4.3 Running **Apps/demo**

Use the following commands to run the application on Ubuntu 16.04 PC:

```
cd GTISDK/Bin/Linux/x86_64
./demo image <model file> <image filename>

./demo video <model file> <video filename>

./demo camera <model file> 0

./demo slideshow <model file> <image folder name>
```

For example models and data, see **Apps/Data** folder.

Note, if you encounter the following error message:

```
bash: ./demo Permission denied
```

Add executable permissions to the demo applications and try again:

```
chmod +x demo
```

4.4.4 Running **Apps/liteDemo**

Use the following command to run the liteDemo application on PC or embedded platforms:

```
cd GTISDK/Bin/Linux/x86_64
./liteDemo <model file> <image filename>
```

4.4.5 Running **Apps/PipelineDemo**

Use the following command to run the Pipelinedemo application on PC or embedded platform:

```
cd GTISDK/Bin/Linux/x86_64
./pipelinedemo <model file> <image folder>
```

4.4.6 Running Python Application (**Apps/Python/demo.py**)

Python application in this directory uses GTILibrary binding located in ../../Python/Lib/gtilib/.
Before running any python application, install GTI Library binding with the following commands:

```
pip install --user ./Python/Lib/gtilib/
```

To run the Python applications:

```
cd Apps/Python
python demo.py image <model file> <image filename>
python demo.py video <model file> <video filename>
python demo.py camera <model file> 0
python demo.py slideshow <model file> <image folder name>
```

Note, you can remove the gtilib package, if it's no longer needed:

```
pip uninstall gtilib
```

4.4.7 Running Android Application (**Apps/Android**)

GTI SDK supports the Android platform. Android application located at Apps/Android is recommended to use, as it's' based on the latest APIs and new model format. To build and run the application on your Android device follow these steps:

1) Setup the Android Studio:
    a) Download the source at https://developer.android.com/studio/ and follow the installation instructions to finalize the environment setup.
2) Once Android Studio is installed, open a terminal, navigate into `<Android Studio Installation Root>/bin` and run the following command:
   `./studio.sh`

3) Navigate to **Tools** → **SDK Manager**, click on **SDK Tool** tab, and select to install **NDK** development tools, as shown below:

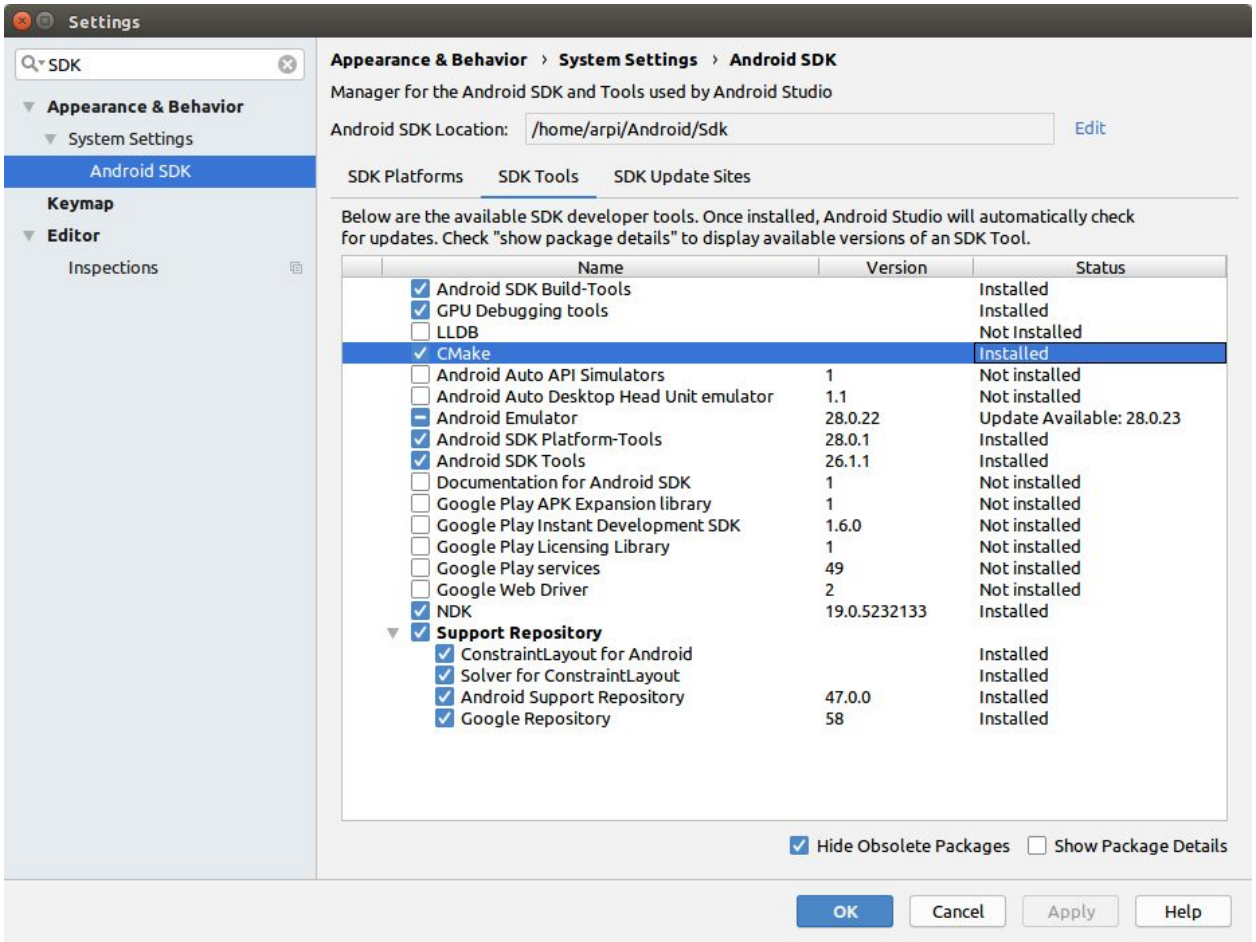

Figure 2. NDK and other tools installed with Android SDK

4) Build your project from the following path "`GTISDK/Apps/Android/jni`":
   `./build.sh`

5) Go back to Android Studio and open the project to create an APK file.
   a) Click **File** then **Open** and select the location of your project, in this example it will be at "`GTISDK/Apps/Android/app/gti4Demo`".
   Note, you may need to add `google()` repository support in your "`app/gti4Demo/build.gradle`" file:

```
allprojects {
    repositories {
        jcenter()
        google()
    }
}
```

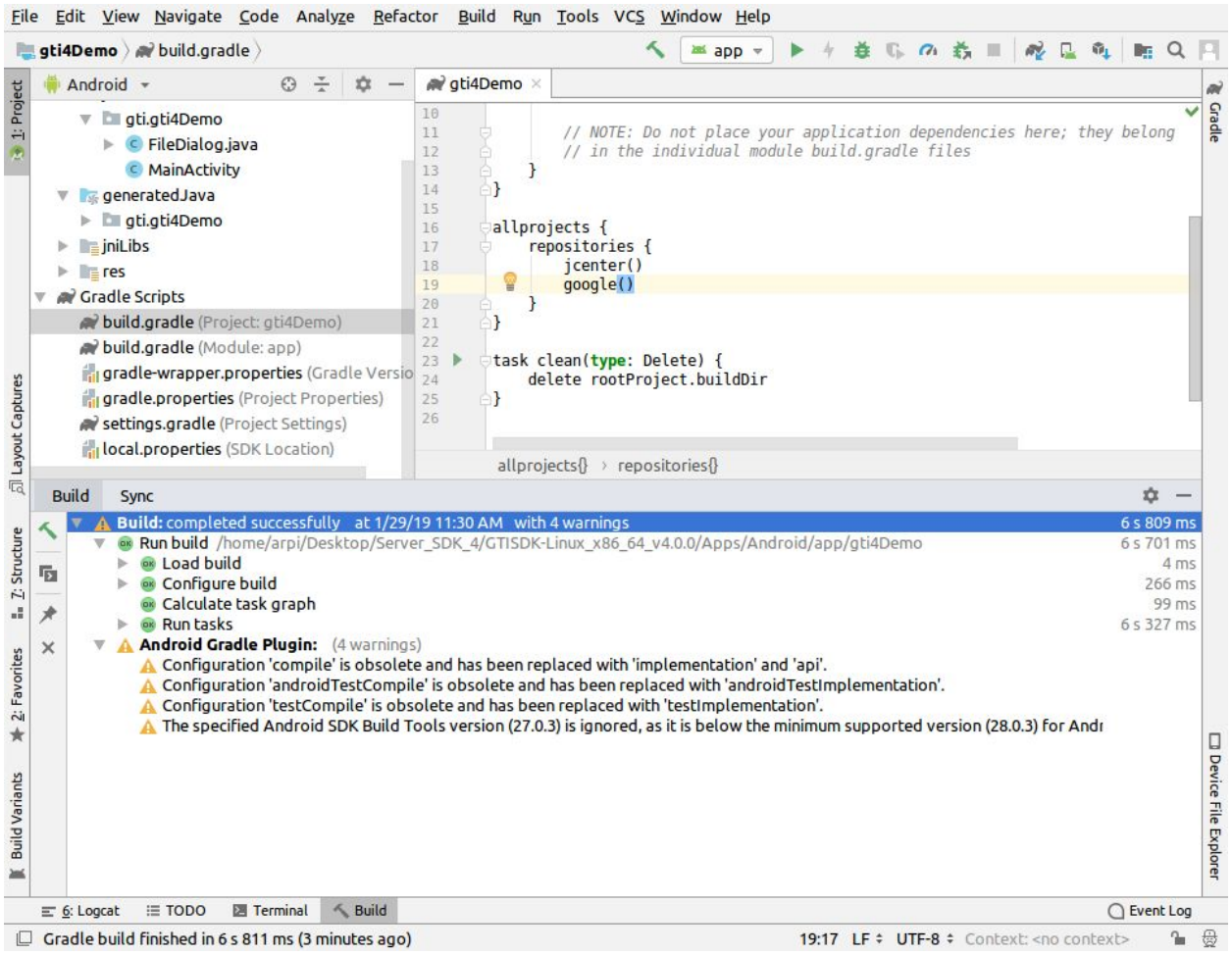

Figure 3. Project is successfully built

b) Click **Build** then **Make Project** to build the project. Once the build is completed successfully, you can generate an unsigned APK.

c) Navigate to **Build** then **Build Bundle (s) /APKs** and select the first option - **Build APK (s)**. The output files (app-debug.apk and output.json) will be stored at "`GTISDK/Apps/Android/app/gti4Demo/app/build/outputs/apk/debug`".

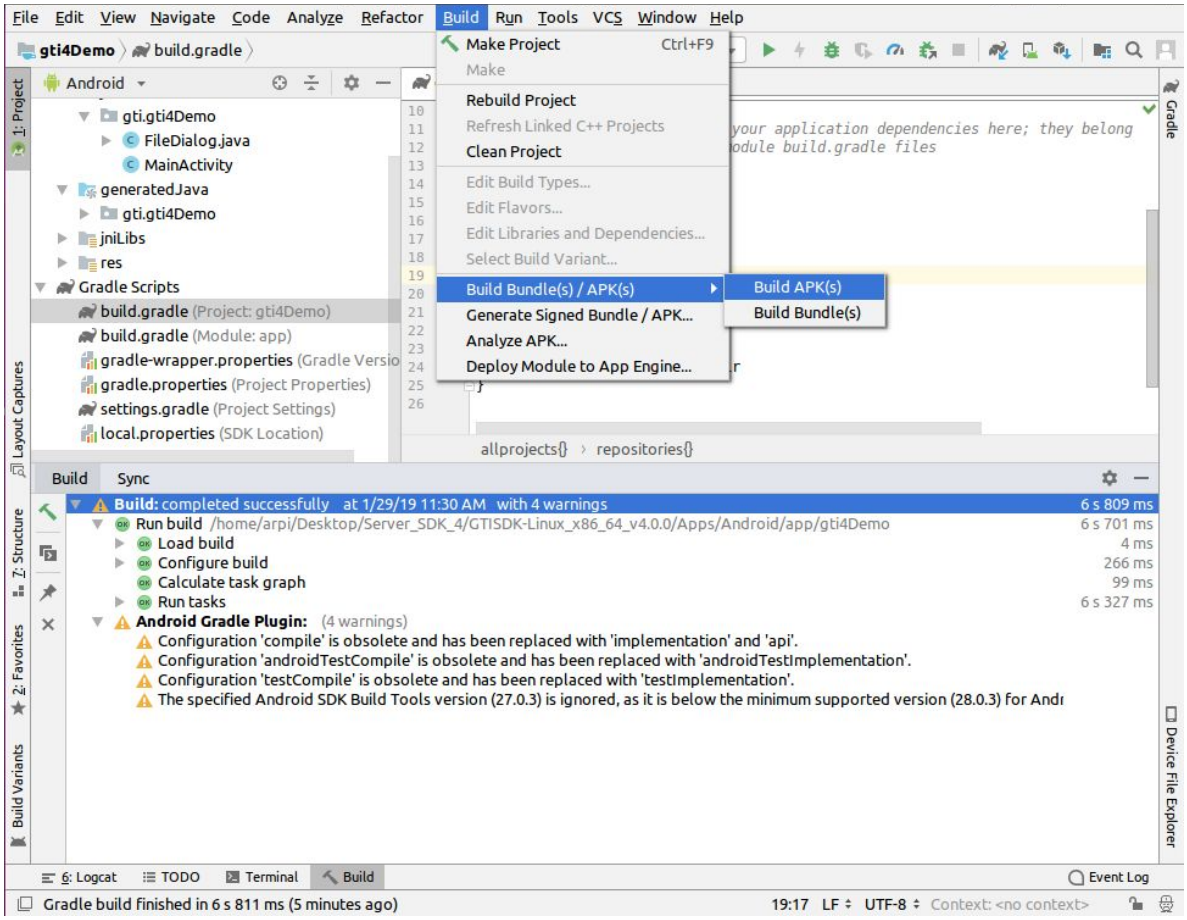

Figure 4. Building your APK

You've now successfully created an APK that can be uploaded and installed on your Android device.

On Android devices:
Note, once the app is installed, grant the necessary permissions for it to function properly (see step 5).

1) Copy the following mandatory supporting libraries (for armeabi-v7a or arm64-v8a architectures) from your PC to the Android device.
    a) libftd3xx.so
    b) libgti_android.so
    c) libgti_gti4Demo.so

2) When connecting to an Android target, the target needs to have USB debug interface enabled and USB interface needs to be under file transfer mode.
3) GTI device mounted on the Android platform must be readable and writable. Note, you can connect to your device using adb shell. Switch to root with "su" then enable read/write with the following command:

    a) For FTDI and DirectUSB dongles:

```
bus=$(lsusb | grep 0403:601 |cut -d' ' -f2)
dev=$(lsusb | grep 0403:601 |cut -d' ' -f4 | cut -d':' -f1)
chmod +rw /dev/bus/usb/${bus}/${dev}  #enable read write
ls -l /dev/bus/usb/${bus}/${dev}  #double check
```

    b) For eUSB dongle:

```
dev=$(($(wc -l /proc/scsi/sg/devices|cut -d' ' -f1)-1))
chmod +rw /dev/sg${dev} #enable read write
ls -l /dev/sg${dev} #confirm that permissions are granted
```

Note, these commands are used every time the dongle is connected to your Android device.

4) The required model and data files should be placed at target Android platform under directory "/sdcard/gti/".

5) Install the APK on your Android device. Android system will prompt to grant permissions. Note that, the prompt window could appear behind the app window, make sure to provide the required permissions. Access to necessary resources should be given for the app to work correctly.

GTI SDK also provides Android application for legacy APIs at "Samples/Android". The installation steps described earlier also apply to this application. Once you've built and installed the APK on the device, launch the application (such as gti4Demo) and allow some time to load the model. After the model is successfully loaded the inferencing will start, as shown below.

The current setup is tested on Android version 7.0 running on Qualcomm Snapdragon μSOM development board. Power up the board, connect to PC with a USB cable, attach GTI 2801 dongle and launch the APK to run CNN inferencing on your data. Inference results are provided on the top of each image.

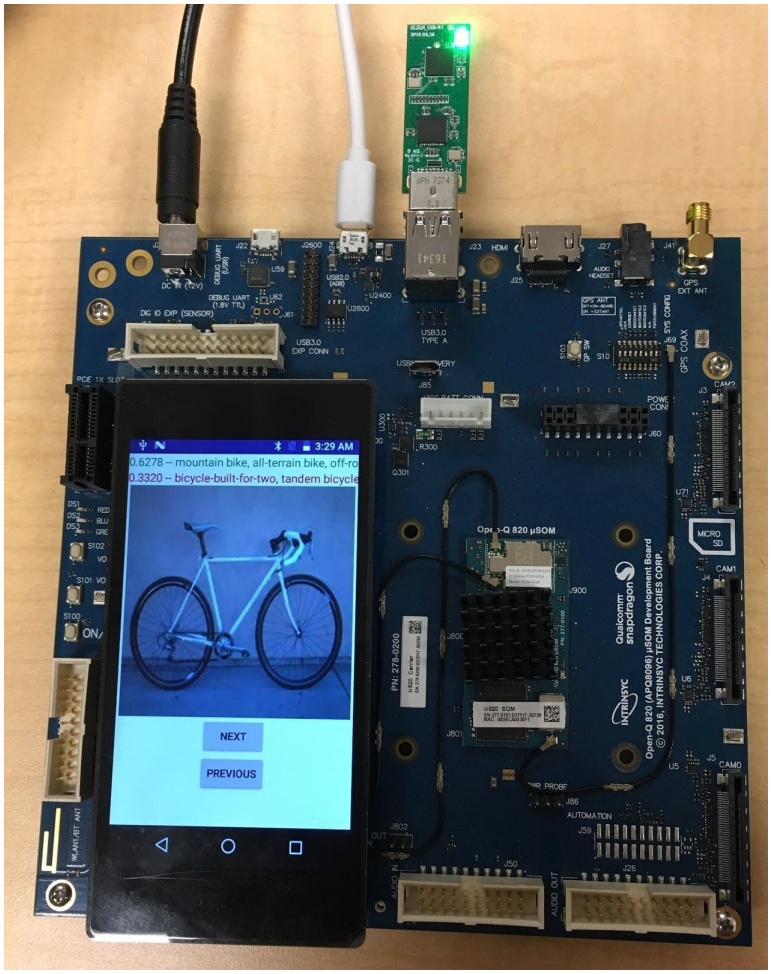

Figure 5. CNN inferencing using gnet1 network model running on GTI 2801 attached to an Android device. Click Next or Previous to navigate through your media files.

4.4.8 Compile Your Application on Linux PC

GTI SDK supports the following interfaces (FTDI and PCIe).

The reference applications with current model format are located under **Apps/Demo** and **Apps/liteDemo**. You can customize and recompile the reference application and copy the executables under to `Bin/Linux/x86_64/.`

```
make -C Apps/Demo
cp -pr Apps/Demo/demo Bin/Linux/x86_64/
```

```
make -C Apps/liteDemo
cp -pr Apps/liteDemo/liteDemo Bin/Linux/x86_64/
```

You can also build applications located under **Samples**:

```
cd GTISDK
make -C Samples/Sample
```

4.4.9 Cross compiling on multiple architectures

Popular embedded architectures are supported through cross compiling from Ubuntu Linux PC including ARM AArch32 and AArch64.

Cross compiling for ARMv7

1) Download Linaro Toolchain from: https://releases.linaro.org/components/toolchain/binaries/latest-6/arm-linux-gnueabihf/ to GTISDK/Build/Tools/.
2) Uncompress the tar file then add **arm gcc** into PATH before building the GTI library and liteSample code.

```
export PATH=/PATH-gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin:$PATH
arm-linux-gnueabihf-gcc -v
```

3) To enable armv7 cross-compiling, run "`source Build/armv7l_cross.env`", as shown below:

```
cd GTISDK
source Build/armv7l_cross.env
CROSS COMPILE for armv7l:
CPU_ARCH   =  armv7l
OS_TYPE    =  Linux
GTI_CC     =  arm-linux-gnueabihf-g++
```

4) Create the armv7 executable file.

```
make -C Samples/Sample lite
make -C Apps/liteDemo
```

Cross compiling for AArch64 (ARMv8a)

1) Download Linaro Toolchain from: https://releases.linaro.org/components/toolchain/binaries/latest-6/ to GTISDK/Build/Tools/.

2) Uncompress the tar file then add **aarch64 gcc** into PATH before building the GTI library and sample code.

```
export PATH=/PATH-gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin:$PATH
aarch64-linux-gnu-g++ -v
```

3) To enable aarch64 cross-compiling, you need to run "source Build/aarch64_cross.env".

```
cd GTISDK
source Build/aarch64_cross.env
CROSS COMPILE for aarch64:
CPU_ARCH   =  aarch64
OS_TYPE    =  Linux
GTI_CC     =  aarch64-linux-gnu-g++
```

4) Create the aarch64 executable file.

```
make -C Samples/Sample lite
make -C Apps/liteDemo
```

If you are using a different toolchain, recompile the sample for your target. You need to modify the .env using your toolchain path and gcc version. The list of toolchains used in SDK 4.0 is shown below:

- gcc-linaro-6.5.0-2018.05-x86_64_aarch64-linux-gnu.tar.gz
- gcc-linaro-6.5.0-2018.05-x86_64_arm-linux-gnueabihf.tar.gz

4.4.10 Image Formatting and Target Configuration Tools

With this release, GTI SDK provides an image conversion tool, model conversion tool and tools to test host to target communication interfaces including FTDI and PCIe.

imageTool

As a prerequisite, install python-opencv.

Run the python script (`gtiConvtImage2BGR_plannar.py`) on the host PC and it will convert an image file to binary in 224x224x3 BGR (default) planar format.

The script runs with the following input parameters.

```
python gtiConvtImage2BGR_plannar.py <inputImage> <image_width> <output_bin>
```

eusbTool

This tool allows testing USB driver and evaluate read/write performance.
Run `./build.sh` to compile from target board.

hwTool

It is a hardware interface test tool to validate data transfer in the chip using various interfaces.

modelTool

Earlier SDK releases use three files to describe a trained neural network - CNN model, fully-connected model, and a class label file. Starting with SDK 4.0, all trained models are described in a single GTI-defined model file.

GTILibrary can now detect the installed GTI hardware, as long as only one type of hardware is installed. In earlier SDK releases, users had to manually edit a configuration file to provide device and connection type, even the duration of data transfer to GTILibrary. In SDK 4.0 this special configuration is now included in the model file itself.

This tool helps to convert old model files into the new GTI model file format or to separate the new GTI model file into old model file format.

Consider gti_gnet3_fc20_2801.model (VGG type model with three fully connected layers and no pooling layer) as an example included in the modelTool folder.

Run the command:

```
./modelTool modeldec gti_gnet3_fc20_2801.model
```

The following 6 files will be generated:

- `gti_gnet3_fc20_2801.model.0` // JSON file that describes the network
- `gti_gnet3_fc20_2801.model.1` // CNN model file
- `gti_gnet3_fc20_2801.model.2` // FC6 model file

- gti_gnet3_fc20_2801.model.3 // FC7 model file
- gti_gnet3_fc20_2801.model.4 // FC8 model file
- gti_gnet3_fc20_2801.model.5 // Label file with 20 classes

Once the model file is decoded, you can customize JSON output file to match your application needs. The full content of the generated JSON file (gti_gnet3_fc20_2801.model.0) is shown below.

```
{
    "data hash": null,
    "layer": [
        {
            "data file": null,
            "data offset": null,
            "input format": "CS_RGB24_PLANAR", # Input channel order
            "name": "image reader",            # Layer Name
            "operation": "IMAGEREADER",   # Layer type
            "output channels": 3,         # Number of output channels(RGB)
            "output format": "CS_BGR24_PLANAR", # Output channel order
            "output height": 224,         # Output dimension - height
            "output width": 224           # Output dimension - width
        },
        # Device configuration file
        {
            "data file": "gti_gnet3_fc20_2801.model.1",
            "data offset": 0,
            "device": {
                "chip": "2801",           # 2801 or 2803
                "emmc delay": 5000,       # for 2803, delay is 12000
                "name": null,
                "type": 0
            },
            "name": "cnn",          # Layer name
            "operation": "GTICNN", # Layer type CNN running on the GTI chip

            "output channels": 256,       # Output channel number
            "output height": 7,           # Output dimension - height
            "output width": 7             # Output dimension - width
        },
        # FC6, FC7, FC8 - three fully connected layers are defined below.
        {
```

```
            "activation": "relu",
            "data file": "gti_gnet3_fc20_2801.model.2",
            "data offset": 0,
            "name": "fc6",
            "operation": "FC",
            "output channels": 128,
            "output height": 1,
            "output width": 1
        },
        {
            "activation": "relu",
            "data file": "gti_gnet3_fc20_2801.model.3",
            "data offset": 0,
            "name": "fc7",
            "operation": "FC",
            "output channels": 128,
            "output height": 1,
            "output width": 1
        },
        {
            "data file": "gti_gnet3_fc20_2801.model.4",
            "data offset": 0,
            "name": "fc8",
            "operation": "FC",
            "output channels": 20,
            "output height": 1,
            "output width": 1
        },
        # Softwax layer runs in the host processor.
        {
            "data file": null,       # No file needed for softmax layer.
            "data offset": null,
            "name": "softmax",
            "operation": "SOFTMAX",
            "output channels": 20,
            "output height": 1,
            "output width": 1
        },

        # File needed by the layer - label file path
        {
```

```
            "data file": "gti_gnet3_fc20_2801.model.5",
            "data offset": 0,
            "name": "label",
            "operation": "LABEL",
            "output channels": 5,
            "output height": 1,
            "output width": 1
        }
    ],
    "name": "gti_2801_gnet3_fc20", # Name of the generated .model file
    "version": "3.5.0"
}
```

The user can also configure the output mode for each layer. Description of each mode is provided below.

MODE 0	FC - fully connected mode, output the last sublayer of the last major layer with each pixel extended from 5 bits to 8 bits by padding three zeros at the LSB side. Note that this padding is only applicable to this mode and not applicable to other modes. And this mode is only applicable to output feature map of size 7x7.
MODE 1	LEARN mode, output all sublayers of all major layers.
MODE 2	SINGLE mode, output the last sublayer of the last major layer for output feature maps of size $\geq$14x14.
MODE 3	SUBLAST mode, output the last sublayers of all major layers.
MODE 4	LASTMAJOR mode, output all sublayers of the last major layer.
MODE 5	LAST_7X7, output the last sublayer of the last major layer for output feature maps of size 7x7.
MODE 6	SUM_7X7, for the last sublayer of the last major layer with output feature map of size 7x7, output the summation of each 7x7 feature map.

Once configuration is complete, run this command to encode the changes back into a single model file:

```
./modelTool modelenc gti_gnet3_fc20_2801.model.0
```

It creates a single GTI model file, based on above 6 files. Rename the file type into .model and this new model file can be used by SDK APIs. All the legacy GTI models can be converted to the new GTI model format with this tool.

5. Windows Support

(To Be Determined)

6. API Guide

The GTI header file includes the GTI SDK library.

6.1 GtiLog Class Reference

Public Member Functions

- std::ostream & GetStream (GTI_LOG_LEVEL level=GTI_LOG_INFO)
- void SetStream (std::ostream *newOs)

Static Public Member Functions

- static void setLevel (GTI_LOG_LEVEL newLevel)
- static GTI_LOG_LEVEL getLevel ()
- static void setLogFile (char *filename)

6.2 GtiTensor Class Reference

Public Attributes:

- int width
- int height
- int depth
- int stride
- void * buffer
- int size
- TENSOR_FORMAT format
- void * tag
- GtiTensor * next
- void * callbackFn

6.3 GtiLib.h File Reference

GTI header file includes the public functions of GTI SDK library.

Classes

- class GtiDevice
- class GtiContext

- class GtiModel
- class GtiTensor

Typedefs

- typedef enum TENSOR_FORMAT GtiModelOutputFormat;
- typedef void(* GtiDeviceNotificationCallBack) (GtiDevice *device)
- typedef void(* GtiEvaluateCallBack) (GtiModel *model, const char *layerName, GtiTensor *output)

Classes

- struct GtiModelProperties{
  char name[FILENAME_STRING_SIZE]; // a null terminated string that holds a name string
  char chipName[FILENAME_STRING_SIZE]; // chip the Model is based on
  };

- struct GtiModelInputInfor{
  int width; // number of input elements in input buffer in horizontal direction
  int height; // number of input elements in input buffer in vertical direction
  int depth; // number of input plane in input buffer
  enum GtiImageColorFormat colorFormat; // color format in the input buffer
  };

- struct GtiModelOutputInfor{
  int width; // number of output elements in output buffer in horizontal direction
  int height; // number of output elements in output buffer in vertical direction
  int depth; // number of output plane in output buffer
  GtiModelOutputFormat format; // tensor element format in the output buffer
  };

- struct GtiModelInfor{
  struct GtiModelProperties modelProp;
  struct GtiModelInputInfor inputInfor;
  struct GtiModelOutputInfor outputInfor;
  };

Enumerations

- enum GTI_DEVICE_STATUS {
  GTI_DEVICE_STATUS_ERROR,
  GTI_DEVICE_STATUS_ADDED,
  GTI_DEVICE_STATUS_REMOVED,
  GTI_DEVICE_STATUS_IDLE,
  GTI_DEVICE_STATUS_LOCKED,
  GTI_DEVICE_STATUS_RUNNING,
  GTI_DEVICE_STATUS_PENDING,
  GTI_DEVICE_STATUS_BUT
  };
- enum GTI_DEVICE_TYPE {
  GTI_DEVICE_TYPE_ALL,
  GTI_DEVICE_USB_FTDI,
  GTI_DEVICE_USB_EUSB,
  GTI_DEVICE_PCIE,
  GTI_DEVICE_VIRTUAL,
  GTI_DEVICE_USB_NATIVE,
  GTI_DEVICE_TYPE_BUT
  };
- enum GTI_CHIP_MODE{
  FC_MODE,
  LEARN_MODE,
  SINGLE_MODE,
  SUBLAST_MODE,
  LASTMAJOR_MODE,
  LAST7x7OUT_MODE,
  SUM7x7OUT_MODE,
  GTI_CHIP_MODE_BUT
  };
- enum TENSOR_FORMAT{
  TENSOR_FORMAT_BINARY, // default is char, set in a model file for CNN layer, with "output type" set as "byte".
  TENSOR_FORMAT_BINARY_INTEGER,
  TENSOR_FORMAT_BINARY_FLOAT, // set in a model file for CNN layer, with "output type" set as "float".
  TENSOR_FORMAT_TEXT,

TENSOR_FORMAT_JSON, // set in a model file for LABEL layer.
TENSOR_FORMAT_UNDEFINED,
TENSOR_FORMAT_BUT,
};

Note, normally TENSOR_FORMAT_BINARY is used (1 byte) for beginning layers, for the last layer, it's either TENSOR_FORMAT_TEXT or TENSOR_FORMAT_JSON.

- enum GtiImageColorFormat {
  GTI_CF_BGR24_PLANAR,
  GTI_CF_RGB24_PLANAR,
  GTI_CF_UNDEFINED,
  GTI_CF_BUT,
  };

6.4 Table of API Functions

GtiChangeModelMode	Change the mode of the device's loaded model.
GtiChangeModelNetworkId	Change the network ID of the device's current model.
GtiCheckDeviceStatus	Get the device's status.
GtiCloseDevice	Close the device.
GtiComposeModelFile	Compose individual layers into a model file.
GtiCreateModel	Create a GTI Model object from a Model File.
GtiCreateModelFromBuffer	Create a GTI Model object from a Model buffer in the memory.
GtiDecomposeModelFile	Decompose a model file into per layer model definition files.
GtiDestroyModel	Destroy a GTI Model object.
GtiDeviceCreate	Create a device object that represents a hardware device chip.
GtiDeviceRelease	Releases GTI chip and memory resource.

GtiEvaluate	Evaluate an input GtiTensor object.
GtiEvaluateWithCallback	Evaluate an input GtiTensor object with a callback.
GtiGetAvailableDevice	Get an available GtiDevice object from the list of unused devices that satisfies the required type.
GtiGetContext	Returns a point to the current GtiContext object.
GtiGetDevice	Get an available GtiDevice object that have required platform name (such as 2801).
GtiGetDeviceName	Get the name of the device in a string.
GtiGetDevicePlatformName	Get the device's platform name in a string.
GtiGetDeviceType	Get the given device's type definition.
GtiGetOutputData	Get the output data from a device as fixed point data.
GtiGetOutputDataFloat	Get the output data from a device as floating point data.
GtiGetOutputLength	Returns a device's output data length.
GtiGetSDKVersion	Returns GTI SDK version as a string.
GtiHandleOneFrame	Sends a 224x224x3 byte image data buffer (b,g,r 3 channels) to GTI2801 and extracts the result in fixed point 32-bit data format.
GtiHandleOneFrameFloat	Sends a 224x224x3 byte image data buffer (b,g,r 3 channels) to GTI2801 and extracts the result in floating point 32-bit data format.
GtiImageEvaluate	Evaluate an input image object presented as an array of bytes.
GtiInitialization	Initializes a device’s running environment.
GtiLoadModel	Load a model to a device directly.
GtiOpenDevice	Open the device for parameter setting and data transferring.

GtiRegisterDeviceEventCallBack	Registers an event callback function of type void func(GtiDevice * device).
GtiRemoveDeviceEventCallBack	Unregisters an event callback function of type void func(GtiDevice * device).
GtiResetDevice	Try to reset the device.
GtiSelectNetwork	Select which network to use in a device.
GtiSendImage	Send an image data buffer to a device with data in fixed point 8-bit format.
GtiSendImageFloat	Send an image data buffer to a device with data in floating point format.
GtiSendTiledImage	Send a tiled image data buffer to a device.
GtiUnlockDevice	Release the device and set the device state free.
GtiGetInforFromModelFile	Get information of a model, including its designated device and input and output properties.
GtiGetInforFromModelBuffer	Get information of a model, including its designated device and input and output properties.

6.5 Description of APIs

6.5.1 GtiGetInforFromModelFile()

Syntax:
```
GTISDK_API int GtiGetInforFromModelFile(const char *modelFile, GtiModelInfor *modelInfor);
```

Description: Get information of a model, including its designated device and input and output properties.

Parameters:

- [in] modelFile -- A constant char string containing the name of the model file
- [out] modelInfor -- Information of this model

Returns:

Zero if the model information is not available; non-zero otherwise

6.5.2 GtiGetInforFromModelBuffer()

Syntax: `GTISDK_API int GtiGetInforFromModelBuffer(const void *modelBuffer, int modelSize, GtiModelInfor *modelInfor);`

Description: Get information of a model, including its designated device and input and output properties.

Parameters:

- [in] modelBuffer -- A constant buffer containing a complete model
- [in] modelSize -- An integer containing the model's size in bytes
- [out] modelInfor -- Information of this model

Returns:
Zero if the model information is not available; non-zero otherwise

6.5.3 GtiCreateModel()

Syntax: `GTISDK_API GtiModel *GtiCreateModel(const char *modelFile);`

Description: Create a GTI Model object from a model File.

Parameters:

- [in] modelFile -- A constant char string containing the name of the model file

Returns:
A pointer to the created GtiModel object; null in case of errors

6.5.4 GtiCreateModelFromBuffer()

Syntax: `GTISDK_API GtiModel *GtiCreateModelFromBuffer(void *modelBuffer, int size);`

Description: Create a GTI Model object given a buffer that contains the entire model.

Parameters:

- [in] modelBuffer -- A pointer to a buffer containing a model as a byte array
- [in] modelSize -- An integer containing the model's size in bytes

Returns:
A pointer to the created GtiModel object; null in case of errors

6.5.5 GtiDestroyModel()

Syntax: `GTISDK_API int GtiDestroyModel(GtiModel *model);`

Description: Destroys a GTI Model object

Parameters:

- [in] model -- A pointer to a GtiModel object

Returns:

6.5.6 GtiEvaluateWithCallback()

Syntax: `GTISDK_API int GtiEvaluateWithCallback(GtiModel *model,GtiTensor * input, GtiEvaluateCallBack fn);`

Description: Evaluate an input GtiTensor object with a callback function. This function sends input data as a GtiTensor object to the established model. The model filters the data through all layers in the network, and call the callback function after the last layer.

Parameters:

- [in] model -- A pointer to the model object
- [in] input -- A pointer to the input GtiTensor object
- [in] fn -- A pointer to the callback function

Returns:
Always 1.

6.5.7 GtiEvaluate()

Syntax: `GTISDK_API GtiTensor * GtiEvaluate(GtiModel *model,GtiTensor * input);`

Description: Evaluate an input GtiTensor object.This function sends input data as a GtiTensor object to the established model. The model filters the data through all layers in the network, and output the results as a GtiTensor object.

Parameters:

- [in] model -- A pointer to the model object
- [in] input -- A pointer to the input GtiTensor object

Returns:
A pointer to the output GtiTensor object

6.5.8 GtiImageEvaluate()

Syntax: `GTISDK_API const char * GtiImageEvaluate(GtiModel *model, const char *image, int height, int width, int depth);`

Description: Evaluate an input image object presented as an array of bytes. This function sends input data as an image arranged in an array of bytes to the established model. The model filters the data through all layers in the network, and output the results as an array of bytes.

Parameters:

- [in] model -- a pointer to a model object
- [in] image -- a char string, the input image file
- [in] height -- height of image in number of pixels
- [in] width -- width of image in number of pixels
- [in] depth -- depth of image in number of channels per pixel

Returns:
A pointer to the output data as an array of bytes

6.5.9 GtiDecomposeModelFile()

Syntax: `GTISDK_API void GtiDecomposeModelFile(const char *modelFile);`

Description: Decompose a model file into per layer model definition files. A model file represents a complete network that consists of multiple layers. This function breaks the model file into multiple individual files each contains definition of a layer. A separate json file that contains the decomposed structure of the network is also generated.

Parameters:

- [in] modelFile -- A constant char string containing the name of a model file

Returns:
None

6.5.10 GtiComposeModelFile()

Syntax: `GTISDK_API void GtiComposeModelFile(const char *jsonFile, const char *modelFile);`

Description: Compose individual layers into a model file. Given a network structure definition defined with a json file, this function assembles individual layer definitions into a complete model

file. Each layer is stored in a file with its filename recorded in the json file. The output is a new model file that consists of all layers.

Parameters:

- [in] jsonFile -- A constant char string for the name of the json file that contains the network structure
- [in] modelFile -- A constant char string containing the name of a model file

Returns:
None

6.5.11 GtiGetSDKVersion()

Syntax: `GTISDK_API const char * GtiGetSDKVersion();`

Description: This function returns GTI SDK version as a string.

Returns:
GTI SDK version

6.5.12 GtiGetContext()

Syntax: `GTISDK_API GtiContext *GtiGetContext();`

Description: This function returns a pointer to the current GtiContext object

Returns:
GTI Context object

6.5.13 GtiRegisterDeviceEventCallBack()

Syntax: `GTISDK_API void GtiRegisterDeviceEventCallBack(GtiContext * context, GtiDeviceNotificationCallBack fn);`

Description: This function registers an event callback function of type void func(GtiDevice * device)

Parameters:

- [in,out] context -- The GtiContext to register event with
- [in] fn -- A function pointer to an event callback function of type void func(GtiDevice * device)

Returns
None

6.5.14 GtiRemoveDeviceEventCallBack()

Syntax: `GtiRemoveDeviceEventCallBack(GtiContext * context, GtiDeviceNotificationCallBack fn);`

Description: This function unregisters an event callback function of type void func(GtiDevice * device)

Parameters:

- [in,out] context -- The GtiContext to unregister event with
- [in] fn -- Pointer to the event callback function

Returns:
None

6.5.15 GtiGetDevice()

Syntax: `GTISDK_API GtiDevice *GtiGetDevice(GtiContext * context, const char *devicePlatformName);`

Description: Get an available GtiDevice object that has the required platform name

Parameters:

- [in] context -- The GtiContext object that contains the GtiDevice
- [in] devicePlatformName -- A string representing the platform's name of a device

Returns
A pointer to the available GtiDevice object

6.5.16 GtiGetAvailableDevice()

Syntax: `GTISDK_API GtiDevice *GtiGetAvailableDevice(GtiContext * context, GTI_DEVICE_TYPE deviceType);`

Description: Get an available GtiDevice object from the list of unused devices that satisfies the required type.

Parameters:

- [in] context -- The GtiContext object that has the GtiDevice
- [in] deviceType -- Enumerated device type

Returns
A pointer to the available GtiDevice object

6.5.17 GtiDeviceRead()

Syntax: `GTISDK_API int GtiDeviceRead(GtiDevice *device,unsigned char * buffer, unsigned int length);`

Description: Read data from the device.

Parameters:

- [in] device -- A pointer to the GtiDevice
- [in] buffer -- A pointer to the read buffer
- [in] length -- read buffer length

Returns:
The number of bytes read, minus numbers mean errors

6.5.18 GtiDeviceWrite()

Syntax: `GTISDK_API int GtiDeviceWrite(GtiDevice *device,unsigned char * buffer, unsigned int length);`

Description: Write data to the device.

Parameters:

- [in] device -- A pointer to the GtiDevice
- [in] buffer -- A pointer to the write buffer
- [in] length -- write buffer length

Returns:
The number of bytes wrote, minus numbers mean errors

6.5.19 GtiUnlockDevice()

Syntax: `GTISDK_API int GtiUnlockDevice(GtiDevice *device);`

Description: Release the device and set the device state free.

Parameters:

- [in] device -- A pointer to the GtiDevice to be unlocked

Returns
True, if the device is successfully unlocked; False, if the device cannot be unlocked

6.5.20 GtiResetDevice()

Syntax: `GTISDK_API int GtiResetDevice(GtiDevice *device);`

Description: Try to reset the device

Parameters:

- [in] device -- A pointer to the GtiDevice

Returns:
True, if the device is successfully reset; False, if the device is not reset

6.5.21 GtiGetDeviceType()

Syntax: `GTISDK_API GTI_DEVICE_TYPE GtiGetDeviceType(GtiDevice *device);`

Description: Get the given device's type definition.

Parameters:

- [in] device -- A pointer to the GtiDevice object

Returns:
Enumerated device type

6.5.22 GtiGetDeviceName()

Syntax: `GTISDK_API const char *GtiGetDeviceName(GtiDevice *device);`

Description: Get the name of the device in a string

Parameters:
- [in] device -- A pointer to the GtiDevice object

Returns:
A null terminated string containing the device's name

6.5.23 GtiGetDevicePlatformName()

Syntax: `GTISDK_API const char *GtiGetDevicePlatformName(GtiDevice *device);`

Description: Get the device's platform name in a string

Parameters:
- [in] device -- A pointer to the GtiDevice

Returns:
A null terminated string representing the platform name

6.5.24 GtiCheckDeviceStatus()

Syntax: `GTISDK_API GTI_DEVICE_STATUS GtiCheckDeviceStatus(GtiDevice *device);`

Description: Get the device's status

Parameters:
- [in] device -- A pointer to the GtiDevice object

Returns:

Enumerated device status

6.5.25 GtiLoadModel()

Syntax: `GTISDK_API int GtiLoadModel(GtiDevice *device, const char * modelUrl, GTI_CHIP_MODE mode, int networkId);`

Description: Load a model to a device directly

Parameters:

- [in] device -- A pointer to the GtiDevice object
- [in] modelUrl -- URL path of the model file in a null terminated string
- [in] mode -- chip mode, eusb, ftdi, or others
- [in] networkId -- ID of the network

Returns:
True, if the model is loaded successfully; False, if the model cannot be loaded

6.5.26 GtiChangeModelMode()

Syntax: `GTISDK_API int GtiChangeModelMode(GtiDevice *device, GTI_CHIP_MODE mode);`

Description: Change the mode of the device's loaded model

Parameters:

- [in] device -- A pointer to the GtiDevice object
- [in] mode -- device's mode (chip mode), e.g. eusb, ftdi, or others

Returns:
True, if the mode is changed successfully; False otherwise

6.5.27 GtiChangeModelNetworkId()

Syntax: `GTISDK_API int GtiChangeModelNetworkId(GtiDevice *device, int networkId);`

Description: Change the network ID of the device's current model

Parameters

- [in] device -- A pointer to the GtiDevice object
- [in] networkId -- ID of the network

Returns:
True, if the network ID is changed successfully; False otherwise

6.5.28 GtiHandleOneFrame()

Syntax: `int GtiHandleOneFrame(GtiDevice *Device, unsigned char *Image224Buffer, unsigned int InputLen, unsigned char *OutputBuffer,unsigned int OutputLen)`

Description: This function sends a 224x224x3 byte image data buffer to GTI chip and extracts the result in fixed point 32-bit data format. Image data buffer contains an image in 8 bit per channel planar RGB or BGR depending on model. Usually models trained by Caffe is BGR, otherwise is RGB.

Parameters:

- [in] Device -- A pointer to the device object created by function GtiDeviceCreate
- [in] Image224Buffer -- The image data buffer, a 224x224x3 byte array
- [in] InputLen -- The length in bytes of the image data buffer Image224Buffer
- [out] OutputBuffer -- Pointer to a prepared buffer for the output data
- [in] OutputLen -- Length of the output data in number of output elements

Returns:
1, if the operation completed successfully; 0, if it failed.

6.5.29 GtiHandleOneFrameFloat()

Syntax: `int GtiHandleOneFrameFloat(GtiDevice *Device, unsigned char *Image224Buffer, unsigned int InputLen, float *OutputBuffer, unsigned int OutputLen)`

Description: This function sends a 224x224x3 byte image data buffer to GTI chip and extracts the result in floating point 32-bit data format. Image data buffer contains an image in 8 bit per channel plannar RGB or BGR depending on model. Usually models trained by Caffe is BGR, otherwise is RGB.

Parameters:

- [in] Device -- A pointer to the device object created by function GtiDeviceCreate
- [in] Image224Buffer -- The image data buffer, a 224x224x3 byte array
- [in] InputLen -- The length in bytes of the image data buffer Image224Buffer
- [out] OutputBuffer -- Pointer to a prepared buffer for the output data
- [in] OutputLen -- Length of the output data in number of output elements

Returns:
1, if the operation completed successfully; 0, if it failed.

6.5.30 GtiGetOutputLength()

Syntax: `unsigned int GtiGetOutputLength(GtiDevice *Device)`

Description: This function returns a device's output data length

Parameters

- [in] Device -- A pointer to the device object

Returns:
Output data length in number of elements

6.5.31 GtiDeviceCreate()

Syntax: `GTISDK_API GtiDevice *GtiDeviceCreate(int DeviceType, char *FilterFileName, char *ConfigFileName);`

Description: Create a device object that represents a hardware device chip. This function sets up which port is used to connect to a device, chooses which Gnet type and CNN mode to be used, it also allocates memory for internal use. The created device is of mode "FC_MODE" if mode is not defined by the environment variable "GTI_CHIP_MODE". Filter file is GTI defined .dat coefficient file, for example: gnet32_fc128_20class.bin. Config file is GTI defined ASCII file for storing configuration information, for example: userinput.txt. These files can be found under the data folder.

Parameters:

- [in] DeviceType -- 0=FTDI; 1=EUSB; 2=PCIe
- [in] FilterFileName -- GTI defined .dat coefficient file to configure the network
- [in] ConfigFileName -- GTI defined ASCII file to configure the GTI SDK usage

Returns:
A pointer to the created GtiDevice object

This function is obsolete. Support will be removed.

6.5.32 GtiDeviceRelease()

Syntax: `GTISDK_API void GtiDeviceRelease(GtiDevice *Device);`

Description: This function releases GTI chip and memory resource.

Parameters:
- [in] Device -- A point to the GtiDevice object to be released

Returns:
None
This function is obsolete. Support will be removed.

6.5.33 GtiOpenDevice()

Syntax: `GTISDK_API int GtiOpenDevice(GtiDevice *Device, char *DeviceName);`

Description: Open the device for parameter setting and data transferring

Parameters:
- [in] Device -- A pointer to the device object created by GtiDeviceCreate
- [in] DeviceName -- The name of the hardware device's device handle, e.g. /dev/sg2

Returns:
Always 1

This function is obsolete. Support will be removed.

6.5.34 GtiCloseDevice()

Syntax: `void GtiCloseDevice(GtiDevice *Device)`

Description: Close the device.

Parameters:

- [in] Device -- A pointer to the device object created by GtiDeviceCreate

Returns:
None
This function is obsolete. Support will be removed.

6.5.35 GtiSelectNetwork()

Syntax: `void GtiSelectNetwork(GtiDevice *Device, int NetworkId)`

Description: Select which one to use in a device. This function selects which one to use if multiple networks are loaded. The network ID is the sequence (order) number of network defined in the Network structure file. The NetworkId number starts from 0. In the file, NetworkId is 0, 1, 2, 3, etc.

Parameters:

- [in] Device -- The device created by GtiDeviceCreate
- [in] NetworkId -- The order number of network defined in network structure file

Returns:
None
This function is obsolete. Support will be removed.

6.5.36 GtiInitialization()

Syntax: `int GtiInitialization(GtiDevice *Device)`

Description: Initializes a devices running environment. This function initializes device's SDK library environment. It loads CNN coefficient filter data to GTI chip.

Parameters:

- [in] Device -- A pointer to the device object created by GtiDeviceCreate

Returns:
Always 1
This function is obsolete. Support will be removed.

6.5.37 GtiSendImage()

Syntax: `int GtiSendImage(GtiDevice *Device, unsigned char *Image224Buffer, unsigned int BufferLen)`

Description: Send an image data buffer to a device with data in fixed point 8-bit format. This function sends a 224x224x3 image data buffer to a device (e.g. a GTI chip). Image data buffer contains an image in 8 bit per channel planar RGB or BGR depending on model. Usually models trained by Caffe is BGR, otherwise is RGB.

Parameters:

- [in] Device -- A pointer to the device object created by GtiDeviceCreate
- [in] Image224Buffer -- A pointer to the 224x224x3 fixed point image data buffer
- [in] BufferLen -- Number of elements in the input buffer

Returns:
Always 0.
This function is obsolete. Support will be removed

6.5.38 GtiSendImageFloat()

Syntax: `int GtiSendImageFloat(GtiDevice *Device, float *Image224Buffer, unsigned int BufferLen)`

Description: Send an image data buffer to a device with data in floating point format. This function sends a 224x224x3 image data buffer to a device (e.g. a GTI chip). Image data buffer contains an image in 32 bit per channel planar RGB or BGR depending on model. Usually models trained by Caffe is BGR, otherwise is RGB. In Image224Buffer, each element size is 4 bytes, i.e. sizeof(float).

Parameters:

- [in] Device -- A pointer to the device object created by GtiDeviceCreate
- [in] Image224Buffer -- A pointer to the 224x224x3 floating point image data buffer
- [in] BufferLen -- Number of elements in the input buffer

Returns:
Always 0.
This function is obsolete. Support will be removed

6.5.39 GtiSendTiledImage()

Syntax: `int GtiSendTiledImage(GtiDevice *Device, unsigned char *Image224Buffer, unsigned int BufferLen)`
Description: Send a tiled image data buffer to a device. This function sends a 224x224x3 tiled image data buffer to a device (e.g. a GTI chip). Image data buffer contains an image in 8 bit per channel planar RGB or BGR depending on model. Usually models trained by Caffe is BGR, otherwise is RGB. In Image224Buffer, each channel's pixel order is tile scanned.

Parameters:

- [in] Device -- A pointer to the device object created by GtiDeviceCreate
- [in] Image224Buffer -- A pointer to the 224x224x3 byte tile image data buffer
- [in] BufferLen -- Number of elements in the input buffer

Returns:
Always 0.
This function is obsolete. Support will be removed

6.5.40 GtiGetOutputData()

Syntax: `int GtiGetOutputData(GtiDevice *Device, unsigned char *OutputBuffer, unsigned int BufferLen)`

Description: Get the output data from a device as fixed point data. This function gets output data from a device (e.g. a GTI chip). The output data is 8-bit integer.

Parameters:

- [in] Device -- A pointer to the device object created by GtiDeviceCreate
- [out] OutputBuffer -- A buffer to store the output data in fixed point format
- [in] BufferLen -- Number of elements in the output buffer

Returns:
Always 0
This function is obsolete. Support will be removed

6.5.41 GtiGetOutputDataFloat()

Syntax: `int GtiGetOutputDataFloat(GtiDevice *Device, float *OutputBuffer, unsigned int BufferLen)`

Description: Get the output data from a device as floating point data. This function gets output data from the device (e.g. a GTI chip). The output data is in floating point format. In OutputBuffer each element's size is 4 bytes, i.e. sizeof(float).

Parameters:

- [in] Device -- A pointer to the device object created by GtiDeviceCreate
- [out] OutputBuffer -- A buffer to store the output data in float point format
- [in] BufferLen -- Number of elements in the output buffer

Returns:
Always 0
This function is obsolete. Support will be removed

7. GTI Model File Format and Layer Components

7.1 GTI Model File Format

Model Header in JSON format
Layer 1(binary or text)
Layer 2(binary or text)
….
Layer N(binary or text)

7.2 GTI Model File Header

It contains the parameters of all layers and the model general information.

7.3 GTI Model File Layers

All layers must have the following parameters:

- “operation”: defined by layer’s name
- “name”: (user input)
- “output width”: (user input)
- “output height”: (user input)
- “output channels”: (user input)

7.3.1 Layer IMAGEREADER

Mandatory parameters:

- “operation”: IMAGEREADER

Optional parameters:

- “input format”: CS_RGB_PLANAR/CS_BGR_PLANAR
- “output format”: CS_RGB_PLANAR/CS_BGR_PLANAR

Layer data:

None

7.3.2 Layer GTICNN

Mandatory parameters:

- “operation”: GTICNN

Optional parameters:

- “mode”: 0/1/2/3/4/5/6/7 - each model has a corresponding mode.
- “output type”: float/byte
- “output scaledown”: 0/-7/-6/…./6/7 - performs shift operation on each pixel for feature value (right or left by 7 bits).
- “Device”:
    - “chip”:”2801”/”2803”
    - “name”:(user input GTI device HW name)
    - “eusb delay”: 12000/(user input for delay in milliseconds)

Layer data:

Binary data contains GTI CNN model data.

7.3.3 Layer FC

Mandatory parameters:

- “operation”: FC

Layer data:

Binary data contains FC data.

7.3.4 Layer POOLING

Mandatory parameters:

- “operation”: POOLING

Optional parameters:

- “mode”: center/average/max/sample/top_left

Layer data:

None

7.3.5 Layer SOFTMAX

Mandatory parameters:

- “operation”: SOFTMAX

Layer data:

None

7.3.6 Layer LABEL

Mandatory parameters:

- “operation”: LABEL

Layer data:

Text data with labels

7.3.7 Layer ELTWISE

Mandatory parameters:

- “operation”: ELTWISE
- “mode”: sum/product/max/pass
- “reference layer”: layer name of the reference layer

Layer data:

None

7.3.8 Layer TYPECONVERT

Mandatory parameters:

- “operation”: TYPECONVERT
- “input type”: char/short/integer/float
- “output type”: char/short/integer/float
- “Shift”: right shift bits

Layer data:

None

8. Appendix A : Support for Legacy SDK 3.1

8.1 Install PCIe driver

To install PCIe drivers for SDK v3.1 and earlier releases, run the following shell command from terminal:

```
cp -i
GTISDK/Drivers/Linux/pcie_driver_2801/pcie_driver_dma/etc/udev/rules.d/xdma-udev-comm
and.sh /etc/udev/rules.d/
cp -i
GTISDK/Drivers/Linux/pcie_driver_2801/pcie_driver_dma/etc/udev/rules.d/60-xdma.rules
/etc/udev/rules.d/

cd GTISDK/Drivers/Linux/pcie_driver_2801/pcie_driver_dma/tests
sudo ./load_driver.sh
```

Check if GTI device nodes are successfully created, and create device symbol nodes for the SDK.

```
ls -alt /dev/gti*
    crw------- 1 root root 242, 200 Jun 24 14:35 /dev/gti0-0
    crw------- 1 root root 242, 201 Jun 24 14:35 /dev/gti0-1
    crw------- 1 root root 242, 202 Jun 24 14:35 /dev/gti0-2
    crw------- 1 root root 242, 203 Jun 24 14:35 /dev/gti0-3

sudo ln -s /dev/gti0-0 /dev/gti2800-0
sudo ln -s /dev/gti0-1 /dev/gti2800-1
sudo ln -s /dev/gti0-2 /dev/gti2800-2
sudo ln -s /dev/gti0-3 /dev/gti2800-3

sudo chmod 666 /dev/gti*
```

To check device availability once drivers are installed, run "lsusb" for FTDI and run "lspci" for PCIe interfaces.

8.2 Install the Model Data Package

The model data package includes sample media and model files.

The SDK includes a data package (valid for SDK v3.1 and under) located in the "data" folder that includes a few models for GTI 2801 chip, but a more comprehensive model collection is provided in a separate package. Users can run the following shell command to install the data package into the target board.

For SDK v3.1 and earlier, add SDK path to PATH:

```
sudo mkdir -p /usr/local/GTISDKPATH
cd GTISDK
find data -type f -depth -print | sudo cpio -pdm /usr/local/GTISDKPATH/
echo "export GTISDKPATH=/usr/local/GTISDKPATH" >> ~/.bashrc
bash
echo $GTISDKPATH
    /usr/local/GTISDKPATH
```

The data package that is released separately supports x86_64 and Windows platforms only. The user must install the data package before running **Samples/Sample/cnnSample** and **Apps/Demo** reference applications.

Instructions to install data package **GTI_Legacy_Models.zip** over Ubuntu 16.04 PC:

```
sudo mkdir -p /use/local/GTISDKPATH
sudo unzip GTI_Models.zip -d /usr/local/GTISDKPATH/
echo "export GTISDKPATH=/usr/local/GTISDKPATH" >> ~/.bashrc
bash
echo $GTISDKPATH
    /usr/local/GTISDKPATH
```

8.3 Run the Legacy Pre-built Applications on GTI 2801 / GTI 2803 (on Linux Platforms)

The GTI SDK provides legacy application support at **Samples** from SDK 3.x releases. Legacy model format uses a file `userinput.txt` that requires manual editing.

- **/Samples/Sample/cnnSample** : Reference application for image, video and camera classifications (x86_64 only).
- **/Samples/Sample/liteSample** : Reference application for image classification.

8.3.1 Running Legacy cnnSample

Before running **cnnSample**, install the data package, as described in Install the Model Data Package.

cnnSample uses OpenCV APIs to operate on media files. The SDK package includes a pre-built OpenCV v3.2 library. Following is the command to run the pre-built **cnnSample** on Ubuntu 16.04 PC.

```
cd GTISDK/Bin/Linux
./run-cnn.sh
```

The demo main menu will appear with the following options:

- Option 0 - **Video Hierarchy.** This option reads a video file and classifies the frames real-time by running the CNN's layers on the GTI 2801/2803 chip and all others on the CPU. While the player is running you can pause/resume it with the space bar and press q or ESC to quit the application.
- Option 1 - **Test CNN speed.** This option exercises the GTI 2801/2803 device hardware and provides a measure of its speed (frame rate), but does not perform the back-end host classification processing.
- Option 2 - **Multi Chips.** This option uses four chips to demonstrate four video classifications, simultaneously.
- Option 3 - **Web camera.** Enables system classification of images from a web camera connected to a USB port on the host computer. Plug in a webcam into the host, run the demo application and select menu item 3. The application requires COM port number the camera is connected to.
- Option 4 - **Exit**. Exit the application.

8.3.2 Running Legacy liteSample

The SDK includes a "`run-lite.sh`" script for x86_64, ARMv7 and aarch64. Check the script file (Bin/Linux/run-lite.sh), for more details. Following are the commands to run pre-built liteSample on Ubuntu 16.04 PC.

```
cd GTISDK/Bin/Linux
./run-lite.sh
```

```
    Usage:
    ./liteSample -c CNN_Model -f FC_Model -l Lable_File -i Input_Image_File -u
Network_Config_File -d device
    DeviceName = /dev/sg2, eusbType = 2
    ---- gtiSetParameters: 2048, Total block = 21952
    ---------- Prediction for /usr/local/GTISDKPATH/data/Image_Lite/bridge.bin -----
    0.999955 - 17 : 17 bridge
    4.50576e-05 - 9 : 9 unknown
    1.57612e-10 - 16 : 16 pagoda
    1.85889e-13 - 0 : 0 hotel
    7.35737e-14 - 18 : 18 helicopter
    End of process image
```

Use command arguments to select different CNN models, FC models, test images, network configuration, and hardware devices.

8.4 Compile and Run Your Application on Linux PC

Follow these instructions to use legacy model formats. You can choose one device type (FTDI or PCIe) depending on your platform (see Samples/Sample/GNUmakefile):

- `CFLAGS += -DUSE_EUSB : using EUSB device`
- `CFLAGS += -DUSE_FTDI : using FTDI device`
- `CFLAGS += -DUSE_PCIE : using PCIE device`

After recompiling the reference application, there are two ways of running the reference application on the target: using **run-cnn.sh/run-lite.sh** script from **Bin** folder, or using **cnnSample/liteSample** commands directly. **run-cnn.sh** is for platforms with OpenCV support**.**

- **run-cnn.sh** or **run-lite.sh** script. These scripts include the library path setup, so the user can run the script without configuration.
- **liteSample** native commands. An example is shown below:

  ```
  cd GTISDK
  source Build/armv7l.env
  make -C Samples/Sample lite
  ```

To run the application, run the native commands. In this example armv7l platform is used.

```
cd GTISDK/
export LD_LIBRARY_PATH=$PWD/Lib/Linux/armv7l/:$LD_LIBRARY_PATH
cd Samples/Sample
```

```
liteSample
    Usage:
    ./liteSample -c CNN_Model -f FC_Model -l Lable_File -i Input_Image_File -u
Network_Config_File -d device
    DeviceName = /dev/sg2, eusbType = 2
    ---- gtiSetParameters: 2048, Total block = 21952
    ------ Prediction for /usr/local/GTISDKPATH/data/Image_Lite/bridge.bin ------
    0.999955 - 17 : 17 bridge
    4.50576e-05 - 9 : 9 unknown
    1.57612e-10 - 16 : 16 pagoda
    1.85889e-13 - 0 : 0 hotel
    7.35737e-14 - 18 : 18 helicopter
    End of process image
```

- To run **cnnSample** directly, include the openCV library path.

```
cd GTISDK/
export
LD_LIBRARY_PATH=$PWD/Lib/Linux/OpenCV/x86_64/:$PWD/Lib/Linux/x86_64/:$LD_LIBRARY_PATH
cd Samples/Sample
cnnSample
```

8.5 Legacy Python Applications

GTI SDK provides a Python reference application under **Python/Samples** for the x86_64 platform only. The Python sample is a simple way to demo picture/video/WebCam/2videos classification. To run the Python sample, you have to install the data package. See the details in Installing the Model Data Package.

- `glconfig.py` : global file to define model and date location
- `gtilib.py` : import GTILibrary API
- `gtiClassify.py` : import GTI classify API
- `gtiSamplePic.py` : demo one picture file classification
- `gtiSampleVideo.py` : demo one video file classification
- `gtiSampleWebCam.py` : demo WebCam classification
- `gtiTwoChipsVideo.py` : demo two video file classification

Before running the Python sample, you need to build libGtiClassify.so as follows:

```
cd GTISDK/
make -C Samples/Sample classifylib
```

If you haven't installed the OpenCV3.2 software, you can use GTI pre-built OpenCV library.

```
cd GTISDK/
export
LD_LIBRARY_PATH=$PWD/Lib/Linux/OpenCV/x86_64/:$PWD/Lib/Linux/x86_64/:$LD_LIBRARY_PATH
```

Run the Python reference application:

```
cd GTISDK/Python/Sample
python gtiSamplePic
```