Lightspeeur® SPR2801S
Neural Network Processor
ARM Development Kit
User's Guide

July 2018, Revision 1.1

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.

US Patents Pending. The Gyrfalcon Technology logo is a registered trademark of Gyrfalcon Technology Inc.

© 2018 Gyrfalcon Technology Inc. All rights reserved. All information is subject to change without notice.

Gyrfalcon Technology Inc.

1900 McCarthy Blvd. Milpitas, CA 95035

contact@gyrfalcontech.com

CONFIDENTIAL

TABLE OF CONTENTS

1. Introduction

Gyrfalcon Technology's Lightspeeur® series intelligent neural processors deliver a revolutionary architecture which features low-power, low-cost, and massively convolutional compute capabilities to provide the user with one new approach to implement energy-efficient device design for deep learning application.

GTI2801 ARM Development Kit (ADK) enables the developer to integrate Gyrfalcon Technology's SPR2801S processor into embedded ARM system, to offload the convolutional neural network (CNN) workload from CPU to SPR2801S.

GTI2801 ADK brings together a hardware-accelerated system and a supporting software – performing functions used in video/image recognition, speech recognition, voice authentication, and autonomous vehicle systems (such as cars, trucks, drones, and more).

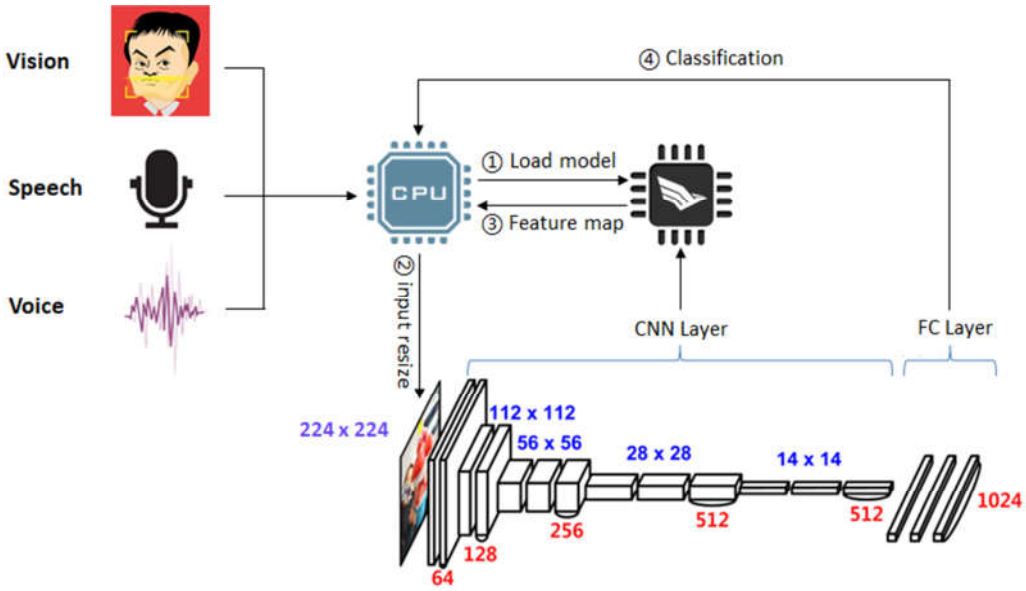


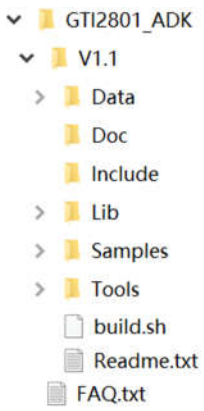
Figure 1. Typical Work Flow Chart of SPR2801S in embedded ARM system

2. GTI2801 ARM Development Kit Installation

2.1 Kit contents

It includes software, documentation and GTI2801 USB Compute Stick:

- Directory of software:



- GTI2801 USB Compute Stick
  - One single, self-contained device; merely plug into a GTI ADK installed ARM platform's USB3.0 (or SS, Super Speed) Type-A port.[1]

    [1] An USB 2.0 port can be used, but with significantly reduced performance versus USB 3.0 port.

2.2 Minimum System Configuration

- An embedded ARM platform with:
  - ARMv7-A or ARMv8-A host CPU processor
  - At least 600MB free storage space of onboard flash
  - 2GB DDR3 @533MHz or higher
  - At least one USB port (USB3.0 is preferred)
  - Linux (Ubuntu 16.04 or Debian 9) or Android 5.x

2.3 Software to be supplied by user

The sample in ADK requires prior installation of OpenCV 3.2.0. The GTI sample uses the video player included in OpenCV to playback and read images from MP4 container files.

2.4 Installation

The package includes the usb driver, sample using OpenCV to handle image, video, and web camera data.

2.4.1 Installing the ADK

```
$ cp  GTI2801_ADK_v1-1.tar.gz  ${WORK_DIR}
$ tar zxvf GTI2801_ADK_v1-1.tar.gz
```

2.4.2 Installing the Opencv3.2.0 [2]

To install the OpenCV, please execute the script as below:

```
$ ./Tools/install_opencv.sh
```

[2] Given ARM application processor with on-chip ISP, image or video functional engines, it is highly recommended to adopt related API invoking specific engine to process camera data, image and video instead of OpenCV by CPU.

2.4.3 Setting the device node

Plug-in the GTI2801 USB Compute Stick, to check which new device node (with name "sg*") is created under path "/dev/" , please execute the command to identify the new device:

```
$ ls –l /dev/sg*
```

```
crw-rw-rw- 1 root disk 21, 0 5月  29 11:25 /dev/sg0
crw-rw-rw- 1 root disk 21, 1 5月  30 14:38 /dev/sg1
crw-rw-rw- 1 root root 21, 2 5月  30 14:38 /dev/sg2
crw-rw-rw- 1 root disk 21, 3 5月  30 15:16 /dev/sg3
```

Figure 2. USB Device node

Then, please configure the file "*Data/Models/gti2801/gnet1/cnn/userinput.txt*", and change the "usb device node" value with the new device node identified in figure 6.

```
{
  "Gti device type": 0,                          # 0: G
  "model": [
    {
        "Network name": "Gnet1",
        "Image output format": 0,          # 0: Conv
        "Dump input image": 0,
        "USB write block numbers": 2048,    # USB dor
        "USB read delay": 12000,        # Delay time
            "USB device node": "/dev/sg0",      # The
}
  ]
}
```

Device node

Figure 3. userinput.txt

*Note:

1- If there is no /dev/sg* identified, you may try to re-compile the kernel to solve, either as below for reference.

- ->make menuconfig

  ->Check the option [*] Enable loadable module support --->

  -> Device Drivers --->

  -> SCSI device support --->

  -> <M> SCSI generic support

  —save—

Then, please re-compile the kernel. The sg.ko will be generated under kernel/drivers/scsi, please copy to ARM platform and execute the command:

$ insmod ./sg.ko

Right now there will be /dev/sg* when you plug-in the USB Compute Stick.

- ->make menuconfig

  ->Check the option [*] Enable loadable module support --->

  -> Device Drivers --->

  -> SCSI device support --->

  -> <*> SCSI generic support

  —save—

Then, please re-compile the kernel. Re-burn the kernel image to ARM platform, and then there will be /dev/sg* when you plug-in the USB Compute Stick.

2.4.4 Installing the usb driver

$ sudo chmod +x build.sh

$ sh build.sh

*Note:

1- Please find under *GTI2801_ADK/V1.1/build.sh* as show in figure 7:

```
if [ -f "/etc/udev/rules.d/50-usb.rules" ]
then
    sudo rm /etc/udev/rules.d/50-usb.rules
fi
sudo cp Lib/Linux/50-usb.rules /etc/udev/rules.d/
sudo udevadm control --reload-rules
```

Figure 4. build.sh

2.4.5 Compiling ADK

Enter into *Samples/Linux/Sample_Gnet1* and execute the following commands to compile ADK,

```
$ make clean;make -j4
```

Then you will get the executable file called ***cnnSample.***

*Note:

1- Make sure use the right makefile, please change file name to "makefile" prior to compiling:
   - *Samples/Sample/makefile_armv7*
   - *Samples/Sample/makefile_armv8*
2- The recommended compiler version are:
   - Gcc-4.8 for ARMv7
   - Gcc-5.4 for ARMv8 (64-bit)

If such error message as below appears:

"libstdc++.so.6: version `GLIBCXX_3.4.21' not found"

Please execute the command:

```
$ cd /usr/lib/arm-linux-gnueabihf
```

Then check whether the version of libstdc++.so.6.0.xx is lower than libstdc++.so.6.0.24, if so, please download the libstdc++.so.6.0.24 and replace libstdc++.so.6.0.xx using the following commands:

```
$ sudo rm libstdc++.so.6.0.xx
$ sudo cp libstdc++.so.6.0.24  /usr/lib/arm-linux-gnueabihf
$ sudo rm libstdc++.so.6
$ sudo ln -s libstdc++.so.6.0.24 libstdc++.so.6
```

3. GTI2801 USB Compute Stick Installation

3.1 GTI 2801 USB Compute Stick Description

GTI2801 USB compute stick is a self-contained device, mounted into a metal enclosure. A powerful, yet surprisingly efficient AI processor SPR2801S embedded into a standard USB stick. The USB compute stick acts as a discrete neural compute accelerator, allowing devices with USB port running neural network at high speed while consuming under a single Watt of power.

Figure 5. Top Side of USB Compute Stick

Figure 6. Back Side of USB Compute Stick

3.2 Hardware installation on Linux

This process assumes that the user has successfully installed the GTI2801 ADK on an Ubuntu based system. Plug the GTI2801 USB Compute Stick into an USB 3.0(or SS) Type-A port of ARM platform.

- A bright green, POWER ON/OK light will illuminate
- The hardware / USB stick is ready for use

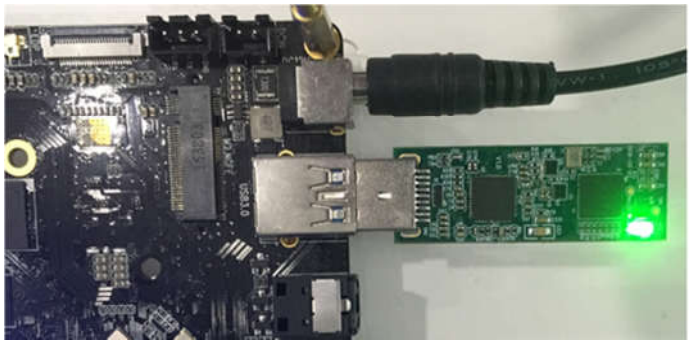


Figure 7. USB Compute Stick plugged into USB3.0/SS port of ARM platform

4. Running the GTI ADK Demo

GTI2801 ADK provides one demo of ILSVRC using standard VGG16 neural network model. After you set up the hardware and software as previously described:

To start the sample, at a Linux command prompt, enter the following command:

```
$ ./cnnSample
```

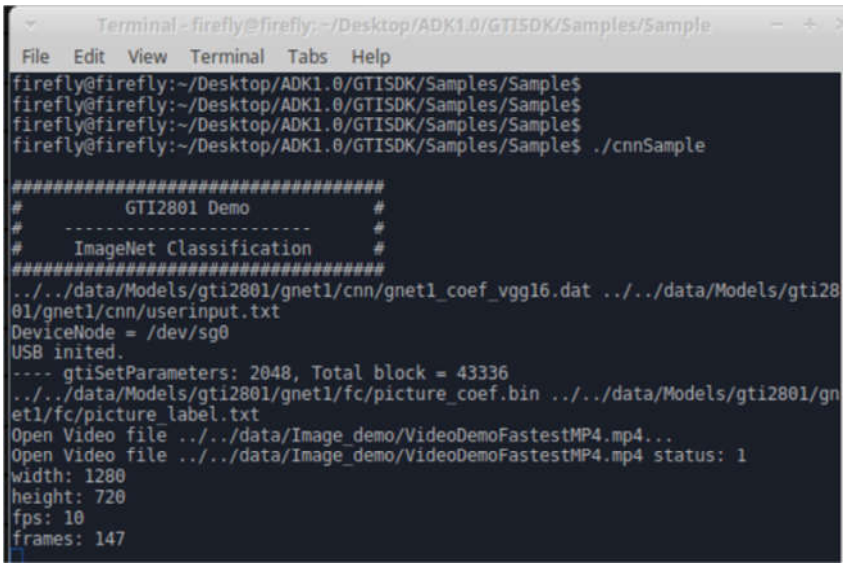


Figure 8. Launch cnnSample

Figure 9. ImageNet Demo

*Note:

1- If such error message as below appears:

bash: ./cnnSample Permission denied

Please enable executable permissions to the demo applications and try again:

```
$ chmod +x cnnSample
$ ./cnnSample
```

2- If such error message as below appears

usb_emmc_write ioctl erro.

Write parameter to eMMC failed.

GTI initialization failed.

Please unplug and re-plugin the USB Compute Stick, then enter the following commands:

```
$ cd /sys/device
$ find -name "max_sectors"
$ su
# cd pci0000:00/0000:00:14.0/usb1/1-7/1-7:1.0/host4/target4:0:0/4:0:0:0
# echo 2048 > "max_sectors"
```

Try again:

```
$ ./cnnSample
```

5. GTI Application Structure

In general, applications call library functions to pass commands, model parameters, and image data to the USB Compute Stick with SPR2801S processor embedded, which outputs its results after processing the commands, model parameters, and data.

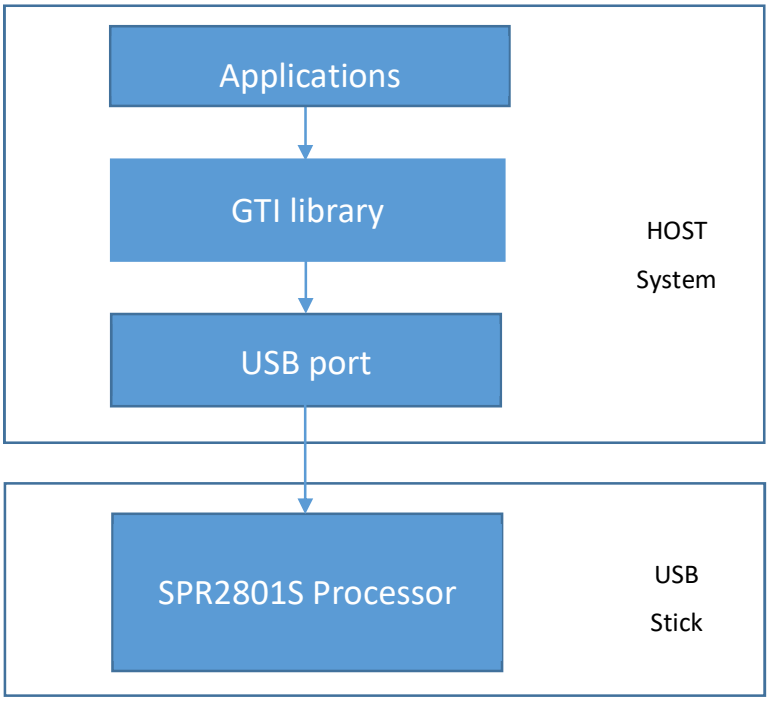


Figure 10. Flow chart of SPR2801S application

6. GTI API LIBRARY

The GTI API library provides functions for applications to initialize state, set command and model parameter registers, transfer image data, and read results from the GTI 2801 system. This library supports basic functionality but does not cover the entire capability of the SPR2802S processor. Although the processor architecture is highly configurable, many of the parameters in the ADK are preset for demonstration purposes.

The API function prototypes are listed in the **GTILib.h** header file. The corresponding precompiled library files are **libGTILibrary-static.a** and **libGTILibrary.so**. The block diagram below describes the basic control flow for the functions used in an application:

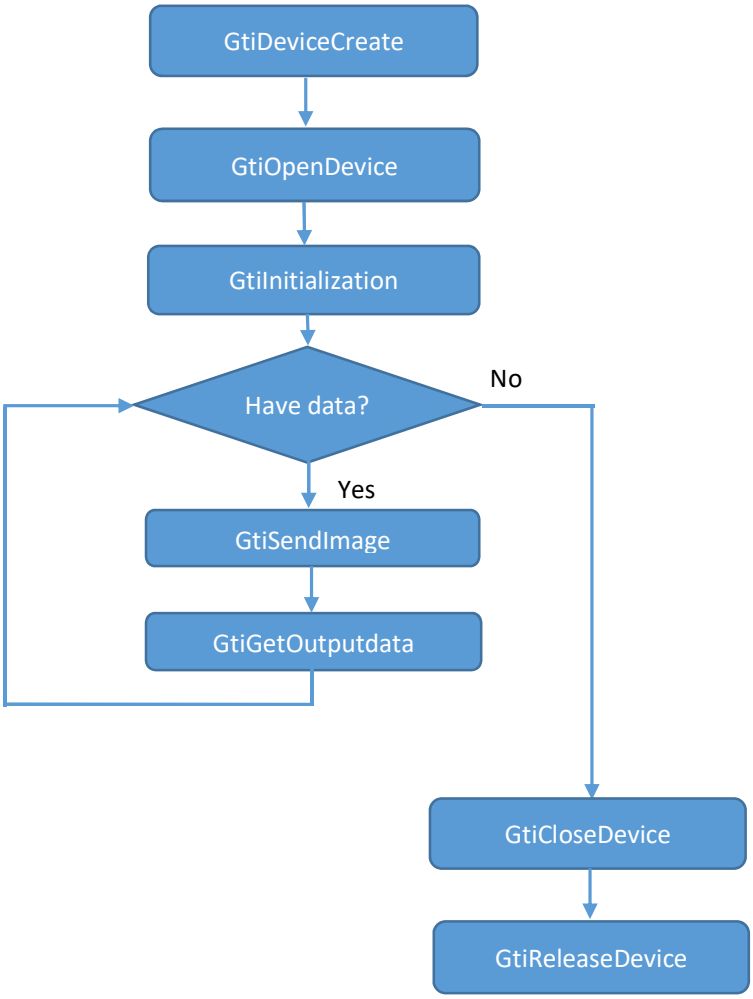


Figure 11. General control flow of software

The following sections describe the API functions in further detail:

6.1 Create GTI device

GtiDevice *GtiDeviceCreate (int PortType, char *FilterFileName, char* ConfigFileName)

This function selects which port is used to connect to the GTI device, loads the chosen network type and CNN model, and allocates memory for internal use.

PortType – specified the port used to connect to the GTI device
0: reserved, 1: usb
FilterFileName – the path to the filter coefficient file
ConfigFileName – the path to the user configuration file
(see below for a description of the format)
Return value – a pointer to the GTI device created

```
# Example user configuration file format:
{
	"Gti device type": 0,
	# 0: GTI 2801
	# 1: GTI 2803
	"model": [
	{
		"Network name": "Gnet1",
		# "Gnet1"
		# "Gnet18"
		"Image output format": 0,
		# 0: Conv out pooling
		# 1: Sub layers
		# 2: Conv out
		# 3: Major layers
	}
	]
}
```

6.2 Open GTI device

int GtiOpenDevice(GtiDevice *Device, char *DeviceName)

This function opens the GTI device for parameter setting and data transferring.

Device – the device created by **GtiDeviceCreate()**
DeviceName – the name for the GTI device connected
Return value – 1: success, 0: failure

6.3 Initialize GTI device

int GtiInitialization(GtiDevice *Device)

This function initializes the GTI library environment.

Device – the device created by **GtiDeviceCreate()**
Return value – 1: success, 0: failure

6.4 Send image to GTI device

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

This function sends the image data to the GTI device. The image data is 224 x 224 pixels, with 3 channels per pixel.

It expects the image to be provided in an unpacked, planar RGB format.

Device – the device created by **GtiDeviceCreate()**
Image224Buffer – the 224x224 pixel, 3-channel *unsigned char* image data buffer
BufferLen – the number of *unsigned char* elements in **Image224Buffer**
Return value – 1: success, 0: failure

6.5 Send Image to GTI device in floating point format

int GtiSendImageFloat(GtiDevice *Device, unsigned char *Image224Buffer, unsigned int BufferLen)
This function sends the image data to the GTI device. The image data is 224 x 224 pixels, with 3 channels per pixel. It expects the image to be provided in an unpacked, planar RGB format.

Device – the device created by **GtiDeviceCreate()**
Image224Buffer – the 224x224 pixel, 3-channel *unsigned char* image data buffer
BufferLen – the number of *unsigned char* elements in **Image224Buffer**
Return value – 1: success, 0: failure

6.6 Get output data length from GTI device

unsigned int GtiGetOutputLength(GtiDevice *Device)
This function gets the output data length from the GTI device.

Device – the device created by **GtiDeviceCreate()**
Return value – the number of elements in the output buffer

6.7 Get output data from GTI device

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

This function gets the output data from the GTI device.

Device – the device created by **GtiDeviceCreate()**
OutputBuffer – the buffer in which to store the output data
BufferLen – the number of *unsigned char* elements in **OutputBuffer**
Return value – 1: success, 0: failure

6.8 Get output data from GTI device in floating point format

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

This function gets the output data from the GTI device in float point format.

Device – the device created by **GtiDeviceCreate()**
OutputBuffer – the buffer in which to store the output data
BufferLen – the number of *float* elements in **OutputBuffer**
Return value – 1: success, 0: failure

6.9 Send image to GTI device and get output data

int GtiHandleOneFrame(GtiDevice *Device, unsigned char *Image224Buffer, unsigned int InputLen, unsigned char *OutputBuffer, unsigned int OutputLen)
This function sends the image data to the GTI device and then gets the output data. The image data is 224 x 224 pixels,

with 3 channels per pixel. It expects the image to be provided in an unpacked, planar RGB format.

Device – the device created by **GtiDeviceCreate()**
Image224Buffer – the 224x224 pixel, 3-channel *unsigned char* image data buffer

InputLen – the number of *unsigned char* elements in **Image224Buffer**
OutputBuffer – the buffer in which to store the output data
OutputLen – the number of *unsigned char* elements in **OutputBuffer**
Return value – 1: success, 0: failure

6.10 Send Image to GTI device and get output data in floating point format

int GtiHandleOneFrameFloat(GtiDevice*Device, unsigned char* Image224Buffer, unsigned int InputLen, float *OutputBuffer, unsigned int OutputLen)
This function sends the image data to the GTI device and then gets the output data. The image data is 224 x 224 pixels, with 3 channels per pixel. It expects the image to be provided in an unpacked, planar RGB format.

Device – the device created by **GtiDeviceCreate()**
Image224Buffer – the 224x224 pixel, 3-channel *unsigned char* image data buffer
InputLen – the number of *unsigned char* elements in **Image224Buffer**
OutputBuffer – the buffer in which to store the output data
OutputLen – the number of *float* elements in **OutputBuffer**
Return value – 1: success, 0: failure

6.11 Close GTI device

void GtiCloseDevice(GtiDevice *Device)
This function closes the GTI device.

Device – the device created by **GtiDeviceCreate()**
No return value

6.12 Release GTI device

void GtiDeviceRelease(GtiDevice *Device)
This function releases the device created by **GtiDeviceCreate()**.

Device – the device created by **GtiDeviceCreate()**

No return value

6.13 Compute the FC Layer

void GtiFc6_mulfloat(float *dst, float *src1, float *src2, int count)
void GtiFc7_mulfloat(float *dst, float *src1, float *src2, int count)
void GtiFc8_mulfloat(float *dst, float *src1, float *src2, int count)

These three functions compute the Fully Connection (FC) layer for classification

No return value

7. GTI ADK Sample Code Description

The C++ source codes include the following files:

GtiClassify.h
Classify_Gnet1.hpp
Classify_Gnet1.cpp
Classify_Gnet18.hpp
Classify_Gnet18.cpp
Samples_Gnet1.hpp
Samples_Gnet1.cpp
Samples_Gnet18.hpp
Samples_Gnet18.cpp

7.1 void convolution(cv::Mat& img)

This function splits the 224x224 pixel image into 3 channels, sends the image data to the GTI device, and gets the output data.

img – the 224 x 224 pixel 3-channel image data
No return value

7.2 Classify(char *CoefDataFcFileName, char *LabelFileName)

This is the Classify construction function that loads the FC coefficient filter and label files and reads the CPU number.

CoefDataFcFileName – coefficient filter FC filename
LabelFleName – the label file name
No return value

7.3 ~Classify()

This is the Classify destruction function that frees the allocated memory from Classify.

No return value

7.4 int GtiClassify(float *InputData, int DataLength, int Count)

This function analyzes the output from the GTI device, classifies the image type, gets a vector formatted as {image name, index}, and returns the sub type.

InputData – the input data for the FC
DataLength – the input data length
Count – the predication label number
Return value – the sub type ID upon success, -1 if the classification failed

7.5 void gtiLoadLabelFile(char *FileName)

This function loads the specified label file.

FileName – the label filename
No return value

7.6 int gtiPreloadCoefData(char *CoefDataFcFileName)

This function loads the specified FC coefficient data file.

CoefDataFileName – the FC coefficient data file name
Return value – 1: success, 0: failure

7.7 float *gtiAccelerateFC(unsigned int *FcOutLength)

This function uses an optimized algorithm to accelerate the FC classification.

FcOutLength – the FC output data length
Return value – the buffer address for accelerated FC result

7.8 vector<int> gtiArgmax(float *InData, unsigned int DataLen, int N)

This function gets the indices of the top N values for the specified vector.

InData – the input data address
DataLen – the input data length
N – the total number of output vector members
Return value – the resulting vector data

7.9 Classify *GtiClassifyCreate(char *CoefDataFcFileName, char *LabelFleName)

This function creates a new Classify object from the specified FC coefficient and label files.

CoefDataFcFileName – the coefficient filter FC filename
LabelFleName – the label filename
Return value – a pointer to the classify object area

7.10 void GtiClassifyRelease(Classify *gtiClassify)

This function releases the Classify object.

gtiClassify – the object created by **GtiClassifyCreate()**
No return value

7.11 int GtiClassifyFC(Classify *gtiClassify, uchar *InputData, int DataLength, int Count)

This function analyzes the output from the GTI device, classifies the image type, gets a vector formatted as {image name, index}, and returns the sub mode type.

gtiClassify – the object created by **GtiClassifyCreate()**
InputData – the input data for the FC
DataLength – the input data length
Count – the predication label number
Return value – the sub mode type ID

7.12 char *GetPredicationString(Classify *gtiClassify, int index)

This function gets the predication string based on the prediction index.

gtiClassify – the object created by **GtiClassifyCreate()**
index – the predication index
Return value – the predication string

7.13 int GetPredicationSize(Classify *gtiClassify)

This function gets the predication vector size.

gtiClassify – the object created by **GtiClassifyCreate()**
Return value – the predication vector size

8. Disclaimer

This disclaimer limits your use of our product. By purchasing our product, you agree to the terms and conditions specified in this agreement. You are held responsible for all system failures regarding the use of Gyrfalcon Technology's SPR2801S AI Processor. Gyrfalcon Technology, Inc. is not responsible for any damage to human life during the demonstration of our product.

This disclaimer releases Gyrfalcon Technology, Inc. from accepting any responsibility to loss of life. You should know that you are using Gyrfalcon Technology's SPR2801S AI Processor at your own risk. You are liable and responsible for any damages. Gyrfalcon Technology, Inc. reserves the right to amend this disclaimer at any time.