Movidius
an Intel company

Movidius™ Neural Compute Stick

API Documentation

July 2017 – Revision 1.1

Movidius
an Intel company

Movidius
an Intel company

Contents

Terminology ... 5
Reference Documents ... 5

1.0 Introduction ... 6
- 1.1 Movidius Neural Compute Workflow ... 6
  - 1.1.1 Neural Network development stage (off NCS device) ... 6
  - 1.1.2 Network compilation & profiling with the Movidius NCS Toolkit ... 7
  - 1.1.3 Product prototyping with NCS to perform CNN acceleration ... 7

2.0 Setup and Installation ... 8
- 2.1 Development model ... 8
- 2.2 Download and installation ... 8
- 2.3 Install Python 3 OpenCV bindings ... 9
- 2.4 Compile the C examples ... 9
- 2.5 Installation on target platforms ... 10
  - 2.5.1 Migration of API to target platform ... 10
  - 2.5.2 Development or deployment of an application on target platform ... 11
- 2.6 API directory structure overview ... 12

3.0 C API ... 13
- 3.1 Enumeration Data Types ... 13
  - 3.1.1 Enum – mvncStatus ... 13
  - 3.1.2 Enum – GraphOptions ... 15
  - 3.1.3 Enum - DeviceOptions ... 16
- 3.2 Functions ... 17
  - 3.2.1 mvncGetDeviceName Function ... 17
  - 3.2.2 mvncOpenDevice Function ... 17
  - 3.2.3 mvncAllocateGraph Function ... 18
  - 3.2.4 mvncDeallocateGraph Function ... 18
  - 3.2.5 mvncLoadTensor function ... 19
  - 3.2.6 mvncGetResult Function ... 20
  - 3.2.7 mvncSetGraphOption Function ... 21
  - 3.2.8 mvncGetGraphOption Function ... 21
  - 3.2.9 mvncSetDeviceOption Function ... 22
  - 3.2.10 mvncGetDeviceOption Function ... 22
  - 3.2.11 mvncCloseDevice function ... 23

4.0 Python API ... 24
- 4.1 Enumerations ... 24
  - 4.1.1 Class Status(Enum) ... 24
  - 4.1.2 Class GlobalOption(Enum) ... 25
  - 4.1.3 Class DeviceOption(Enum) ... 26

4.1.4 Class GraphOption(Enum) ... 26
4.2 Global functions ... 27
4.2.1 EnumerateDevices Function ... 27
4.2.2 SetGlobalOption Function ... 27
4.2.3 GetGlobalOption Function ... 28
4.3 Device Class ... 28
4.3.1 _init_ Method ... 28
4.3.2 OpenDevice Function ... 29
4.3.3 CloseDevice Function ... 29
4.3.4 SetDeviceOption Function ... 29
4.3.5 GetDeviceOption Function ... 30
4.4 Graph Class ... 30
4.4.1 AllocateGraph Function ... 30
4.4.2 DeallocateGraph Function ... 31
4.4.3 SetGraphOption Function ... 31
4.4.4 GetGraphOption Function ... 32
4.4.5 LoadTensor Function ... 32
4.4.6 GetResult Function ... 33
5.0 Examples ... 34
5.1 C Examples ... 34
5.2 Python Examples ... 35

Movidius
an Intel company

Revision History

Date	Revision	Description
July 2017	1.0	Initial release
July 2017	1.1	Added an additional target platform

Note: Review the readme files provided with any software packages for the latest information.

Terminology

The following table provides the meaning of the abbreviations mentioned in this document, as well as some definitions for some specific terms.

Term	Description
API	Application programming interface
Caffe	A deep learning framework used to develop networks that can be compiled to run on the NCS
CNN	Convolutional neural network
Debian®-based Linux* OS	An Operating System (OS) that uses the Linux* kernel and accepts precompiled packages as a way to install user applications.
Host	System that the NCS is connected to
Inference	The act of comparing input to a network knowledge base, whereon a subject's attributes can be inferred
NCS	Neural Compute Stick
NCS SDK	A software package that contains the Toolkit and API for the NCS
VPU	Visual processing unit

Reference Documents

Visit *developer.movidius.com* for additional documentation and information.

1.0 Introduction

This document covers installation of the Movidius™ Neural Compute API (API) on a host system, details of the included API commands, and limited examples of basic functions.

The API provides a lightweight interface enabling developers to initialize a Movidius™ Neural Compute Stick (NCS), load a graph compiled by the Movidius™ Neural Compute Toolkit (referred to as Toolkit), and offload the execution of convolutional neural network (CNN) inferences from a host device.

The API and Toolkit are installed onto an x86-64-based PC developer platform running Ubuntu 16.04. Developers utilize the Toolkit to generate a graph file, and the API to prototype and test an application, both on this development machine.

During API install, Toolkit files are required to compile networks and examples.

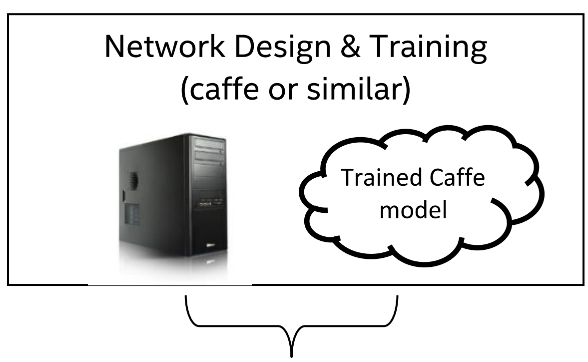
The API also includes redistributable packages to provide C and Python3 code access for all supported target platforms (see release notes), for example on an embedded device. After installing the desired packages on a target, developers can continue application development locally, or use to deploy a final application.

1.1 Movidius Neural Compute Workflow

The following diagram shows a typical conceptual workflow for development and prototyping with the NCS. This workflow uses both components of the Movidius™ Neural Compute SDK – the Toolkit and the API.

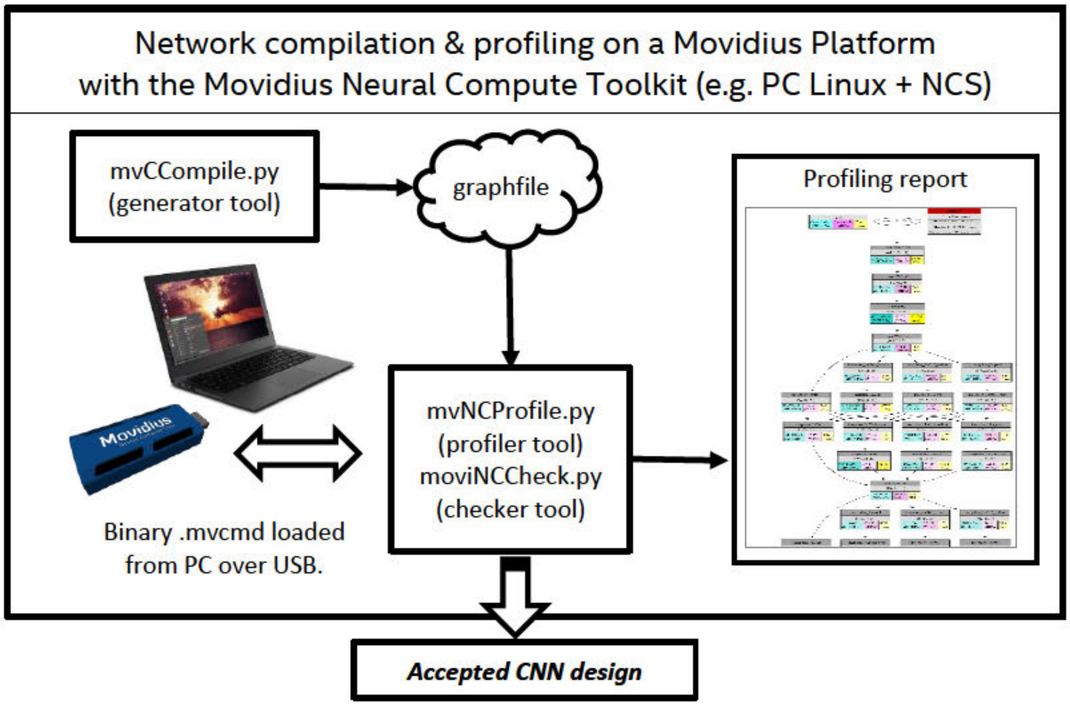
1.1.1 Neural Network development stage (off NCS device)

During this phase, neural networks are designed and trained using appropriate DNN frameworks, typically performed on server or cloud equipment. This process is out of scope for the Movidius SDK and NCS



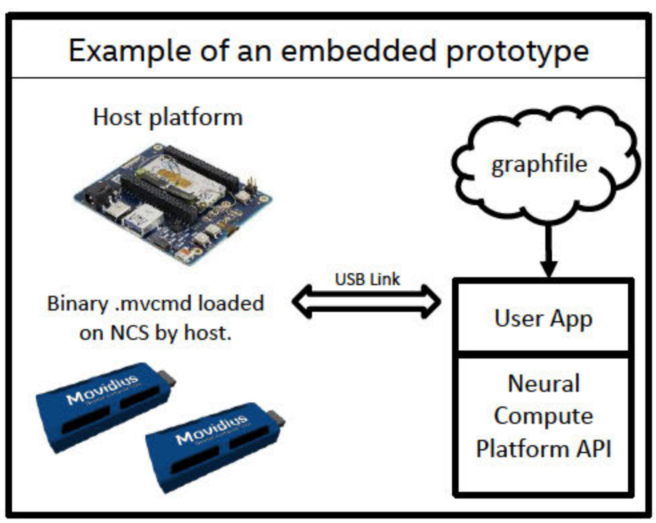
1.1.2 Network compilation & profiling with the Movidius NCS Toolkit

Neural compute toolkit enables users to compile and profile a network, then check a graph against caffe using a single Neural Compute Stick.



1.1.3 Product prototyping with NCS to perform CNN acceleration

Neural compute platform API allows user applications running on host systems to run the network on one, or more, Neural Compute Sticks.



2.0 Setup and Installation

The API is part of the Movidius™ Neural Compute SDK, which also includes the Toolkit.

The Toolkit must be installed before the API, as the Toolkit generates graph files used by the API. See the Movidius™ Neural Compute Toolkit user guide for more information.

2.1 Development model

For development purposes, the API package is intended to be installed on a host computer running Ubuntu 16.04 LTS x86-64 bit alongside the Toolkit.

The API package also contains supporting libraries, as .DEB packages, that enable development and deployment of application for various embedded platforms.

2.2 Download and installation

Download the latest Movidius™ NC SDK package from download area of the user forum at *ncsforum.movidius.com* and review related information.

Successful installation of the API requires that the Toolkit is already installed. Some of the following steps are common with the Toolkit installation steps, and thus the files and directories and may already exist on your system.

Proceed to **Unpack the API archive** if the Toolkit is already installed.

Create a directory for the SDK.

```
$ mkdir <path-to-SDK>
```

Move the SDK archive to the <path-to-SDK> directory.

```
$ mv <MvNC_SDK>.tgz <path-to-SDK>
```

Change directory to the <path-to-SDK> directory.

```
$ cd <path-to-SDK>
```

Unpack the SDK archive.

```
$ tar -xvf <MvNC_SDK>.tgz
```

Note: Ensure the Toolkit has been installed into <path-to-SDK>/bin before continuing to install the API; see the Toolkit user guide.

Note: Before continuing, verify that $PYTHONPATH points to the location that you selected for Caffe during Toolkit installation. If $PYTHONPATH is not defined, execute the following and recheck:

$ source ~/.bashrc

Unpack the API archive.

```
$ tar -xvf <MvNC_API>.tgz
```

After decompression a new directory named ncapi is created. Change directory to the ncapi directory.

```
$ cd ncapi
```

Running the API setup script installs supporting libraries for x86-64, downloads sample networks, and uses the Toolkit to generate graph files.

```
$ ./setup.sh
```

2.3 Install Python 3 OpenCV bindings

In order to run the included Python 3 examples that require cv2, execute the following command:

```
$ ./py_examples/opencv/install_opencv.sh
```

2.4 Compile the C examples

The C examples must be compiled before they can be used:

```
$ cd c_examples && make
```

2.5 Installation on target platforms

After initial installation of the Toolkit and API on the x86-64 development system, and after you have a tested graph file, various flows are possible to migrate your workflow to the target platform, depending on desired outcome.

Manual installation of .DEB packages for the desired target architecture is required. DEB packages are found in <path-to-sdk>/ncapi/redist.

- Packages that end in _amd64.deb are for an x86-64 architecture.
- Packages that end in _armhf.deb are for hard-float ABI ARM architectures, such as Raspberry PI 3 Model B running Raspian Jessie.
- Packages that end in _all.deb are intended for all architectures.

2.5.1 Migration of API to target platform

After successful installation on an x86-64 developer system, copy the entire contents of the <path-to-sdk>ncapi to the target platform. This will ensure the sample networks and supporting files used by the examples will be downloaded and converted.

Note: The toolkit is not currently supported on target platforms.

If you have previously compiled C examples on the development machine, it is a best practice to clean these binaries since they will not operate if the target platform architecture is different than your development platform.

Note: For some API examples to run, OpenCV must first be installed on the target platform.

Follow the instructions from your OS vendor to install OpenCV (with Python3 support). After installing OpenCV, use the included install_opencv.sh script for Ubuntu, or an alternate method compatible with your OS, to install Python3 OpenCV bindings.

Note: For some API examples to run, gstreamer Python3 bindings must be installed on the target platform. For Raspberry PI Raspian Jessie, install using

```
$ sudo apt install python3-gst-1.0
```

2.5.2 Development or deployment of an application on target platform

If you have a specific development activity in mind, or simply wish to deploy a finished application, you may wish to merely copy your graph file, application code, and API redistributable .deb packages to the target platform. Install the specific .DEB packages from the correct subdirectory for your hardware architecture.

An example of deployment for Raspberry PI Raspian Jessie follows:

Copy DEB packages from redistributable directory to removable media. mvnc-dev*.deb and python3-mvnc*.deb are optional depending on if you will be doing native development or running Python on the target platform.

```
        $ cp -R <path-to-SDK>/ncapi/redist/pi_jessie /media/<user
name>/<media label>
```

Copy DEB packages from removable media to Raspberry PI

```
        $ cp -R /media/<username>/<media label>/pi_jessie ~/.
```

Install DEB packages on Raspberry PI

```
        $ sudo dpkg -i ~/pi_jessie/*.deb
```

Copy application source code, and any supporting files such as compiled graph file, to Raspberry PI and then build and/or test the application.

2.6 API directory structure overview

Directory	Purpose
<path-to-SDK>\bin	Toolkit directory, from toolkit installation
<path-to-SDK>\ncapi	API directory.
<path-to-SDK>\ncapi\c_examples	Source and makefile for C examples **ncs-check** **ncs-fullcheck** **ncs-threadcheck**
<path-to-SDK>\ncapi\c_examples\LICENSE	Third party licenses
<path-to-SDK>\ncapi\networks	Well-known example networks. **Note:** for this version of SDK, the original Prototext for these networks has been modified to make them compatible with the Toolkit.
<path-to-SDK>\ncapi\tools	Installation scripts, and destination for **synset.words.txt** If this file is not present after install, some examples will not function. Please verify PYTHONPATH points at your Caffe installation and re-run setup.sh
<path-to-SDK>\ncapi\images	Images for use with code examples
<path-to-SDK>\ncapi\mean	Generated by setup.sh, used by some examples.
<path-to-SDK>\ncapi\py_examples	Contains python examples age_gender_example.py **classification_example.py** **ncs_camera.py**
<path-to-SDK>\ncapi\py_examples\ncscamera	Package used by ncs_camera.py
<path-to-SDK>\ncapi\py_examples\stream_infer	Contains python example **stream_infer.py**
<path-to-SDK>\ncapi\redist	Contains deb packages for installation on various target platforms. **mvnc*.deb** is base package, use this when distributing a finished application. **mvnc-dev*.deb** is used when developing C applications on the target **python3-mvnc*.deb** is used to support python3.5 applications.
<path-to-SDK>\ncapi\redist\pi_jessie	Contains deb packages for installation on Raspberry PI 3 Model B. **mvnc*.deb** is base package, use this when distributing a finished application. **mvnc-dev*.deb** is used when developing C applications on the target **python3-mvnc*.deb** is used to support python3.4 applications.

3.0 C API

The API includes a native C API that is comprised of a shared library (libmvnc.so) and header file (mvnc.h) that provide access to the features of the NCS from a C or C++ program.

3.1 Enumeration Data Types

3.1.1 Enum – mvncStatus

mvncStatus is an enumerated data type that defines the status code returned from most calls to the API library functions. The possible status codes are shown below.

```
enum mvncStatus{
  MVNC_OK = 0,
  MVNC_BUSY = -1,
  MVNC_ERROR = -2,
  MVNC_OUT_OF_MEMORY = -3,
  MVNC_DEVICE_NOT_FOUND = -4,
  MVNC_INVALID_PARAMETERS = -5,
  MVNC_TIMEOUT = -6,
  MVNC_MVCMDNOTFOUND = -7,
  MVNC_NODATA = -8,
  MVNC_GONE = -9,
  MVNC_UNSUPPORTEDGRAPHFILE = -10
  MVNC_MYRIADERROR = -11

};
```

Enum contants

Constant	Description
MVNC_OK = 0	The function call worked as expected.
MVNC_BUSY = -1	The device is busy, retry later.
MVNC_ERROR = -2	An unexpected error was encountered during the function call.
MVNC_OUT_OF_MEMORY = -3	The host is out of memory.
MVNC_DEVICE_NOT_FOUND = -4	There is no device at the given index or name.
MVNC_INVALID_PARAMETERS = -5	At least one of the given parameters is invalid in the context of the function call.
MVNC_TIMEOUT = -6	Timeout in the communication with the device.
MVNC_MVCMDNOTFOUND = -7	The file named *MvNCAPI.mvcmd* should be installed in the *mvnc* directory. This message means that the installer failed.
MVNC_NODATA = -8	No data to return.
MVNC_GONE = -9	The graph or device has been closed during the operation.
MVNC_UNSUPPORTEDGRAPHFILE = -10	The graph file may have been created with an incompatible prior version of the Toolkit. Try to recompile the graph file with the version of the Toolkit that corresponds to the API version.
MVNC_MYRIADERROR=-11	An error has been reported by Movidius™ VPU. Use MVNC_DEBUGINFO.

3.1.2 Enum – GraphOptions

The GraphOptions enumeration is a set of pre-defined values that represent options for the graph. The GraphOptions enumeration is used with the mvncGetGraphOption and mvncSetGraphOption functions.

```
enum GraphOptions{
  MVNC_DONTBLOCK = 2,
  MVNC_TIMETAKEN = 1000,
  MVNC_DEBUGINFO = 1001
}
```

Enum Constants

Constant	Data type	Possible values	Option type	Description
MVNC_ DONTBLOCK = 2	boolean	0 (default = 1)	set/get	0: Calls to LoadTensor and GetResult block. 1: Calls to LoadTensor return BUSY, calls to GetResult return NODATA.
MVNC_ TIMETAKEN = 1000	float*	Time in seconds	get	Time taken for the last inference returned by GetResult.
MVNC_ DEBUGINFO = 1001	string	Debug information	get	Present if the previous error was MYRIADERROR.

3.1.3 Enum - DeviceOptions

GET ORIGINATL FROM MOVIDIUS DOC

The DeviceOptions enumeration is a set of pre-defined values that represent options for the device. The DeviceOptions are used with the mvncSetDeviceOption and mvncGetDeviceOption functions.

```
enum DeviceOptions{
  MVNC_LOGLEVEL = 0,
}
```

Enum constants

Constant	Data Type	Possible values	Option type	Description
MVNC_LOGLEVEL = 0	Int	0 = nothing (default), 1 = errors, 2 = verbose	get/set	Log level
THERMAL_THROTTLING_LEVEL = 1002	Int	Returns 1 if lower guard temperature threshold of chip sensor is reached. This indicates short throttling time is in action between inferences to protect the device. Returns 2 if upper guard temperature of chip sensor is reached. This indicates long throttling time is in action between inferences to protect the device.	get	Throttling level

3.2 Functions

3.2.1 mvncGetDeviceName Function

This function is used to get the device name. To identify all the NCS devices in the system, the user should call this function multiple times while incrementing the index until an error is returned.

```
mvncStatus mvncGetDeviceName(int index, char *name, unsigned
int nameSize);
```

Arguments

Name	Type	Description
index	int	Zero-based index of the device for which a name will be returned.
name	char*	Pointer to the buffer used to store the name of the device.
nameSize	unsigned int	Size in bytes of the buffer pointed to by the name parameter.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.2 mvncOpenDevice Function

This function is used to initialize the device.

```
mvncStatus mvncOpenDevice(const char *name, void
**deviceHandle);
```

Arguments

Name	Type	Description
name	const char*	Pointer to a constant array of chars that contains the name of the device to open. This value is obtained from mvncGetDeviceName.
deviceHandle	void**	Address of a pointer that will be set to point to an NCS device.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.3 mvncAllocateGraph Function

This function allocates a graph on the device and creates a handle to the graph which can be passed to other API function calls such as mvncLoadTensor and mvncGetResult.

```
mvncStatus mvncAllocateGraph(void *deviceHandle, void
**graphHandle, const void *graphFile, unsigned int
graphFileLength);
```

Arguments

Name	Type	Description
deviceHandle	void*	Pointer obtained from a previous call to mvncOpenDevice() that specifies the NCS device to access.
graphHandle	void**	Address of a pointer that will be set to point to a graph upon successful return. The graph is an opaque format. This format can be passed to other API functions that require a graphHandle.
graphFile	const void*	Pointer to a buffer that contains the content of a graph file. Graph files can be created via the Toolkit.
graphFileLength	unsigned int	Length in bytes of the buffer pointed to by a graphFile parameter.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.4 mvncDeallocateGraph Function

This function is used to deallocate a graph on the device. This is a reserved call and may not be implemented in all versions.

```
mvncStatus mvncDeallocateGraph(void *graphHandle);
```

Arguments

Name	Type	Description
graphHandle	void*	Pointer to the opaque graph structure. This pointer should be initialized via a call to the mvncAllocateGraph function.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.5 mvncLoadTensor function

This function is used to initiate an inference on the specified graph via the associated NCS device.

```
mvncStatus mvncLoadTensor(void *graphHandle, const void
*inputTensor, unsigned int inputTensorLength, void *userParam);
```

Arguments

Name	Type	Description
graphHandle	void*	Pointer to the opaque graph structure. This pointer should be initialized via a call to the mvncAllocateGraph function prior to calling this function.
inputTensor	const void*	Pointer to tensor data buffer which contains 16 bit half precision floats (per IEEE 754 half precision binary floating-point format: binary16). The values in the buffer are dependent on the CNN (graph).
inputTensorLength	unsigned int	Length in bytes of the buffer pointed to by the inputTensor parameter.
userParam	void*	Pointer to the user parameter that is returned in mvncGetResult along with the inference result for this tensor.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.6 mvncGetResult Function

This function receives the result of the graph processing. This function blocks according to the value of the GraphOption MVNC_DONT_BLOCK. If not blocking it will return MVNC_NODATA when there is no inference result to return.

```
mvncStatus mvncGetResult(void *graphHandle, void **outputData,
unsigned int *outputDataLength, void **userParam);
```

Arguments

Name	Type	Description
graphHandle	void*	Pointer to the opaque graph structure. This pointer is initialized via a call to the mvncAllocateGraph function.
outputData	void**	Address of the pointer that will be set to point to a buffer of 16 bit floats which contain the result of an inference. The buffer will contain one 16 bit float for each network category. The values are the results of the output node.
outputDataLength	unsigned int*	Pointer to an integer that will be set to the number of bytes in the outputData buffer.
userParam	void**	Address of a pointer that will be set to point to the user parameter for this inference as passed to mvncLoadTensor.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.7 mvncSetGraphOption Function

This function is used to set an option of the graph. The available options can be found in the GraphOptions enumeration.

```
mvncStatus mvncSetGraphOption(void *graphHandle, int option,
const void *data, unsigned int datalength);
```

Arguments

Name	Type	Description
graphHandle	void*	Pointer to the opaque graph structure; initialized via a call to the mvncAllocateGraph function.
option	int	Integer from the GraphOptions enumeration
data	const void*	Pointer to the value of the graph option to set. Type of data will depend on option specified.
datalength	unsigned int	Length in bytes of the value pointed to by the data parameter.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.8 mvncGetGraphOption Function

This function is used to retrieve the optional information from the graph. The available options can be found in the GraphOptions enumeration.

```
mvncStatus mvncGetGraphOption(void *graphHandle, int option,
void **data, unsigned int *datalength);
```

Arguments

Name	Type	Description
graphHandle	void*	Pointer to the opaque graph structure; initialized via a call to the mvncAllocateGraph function.
option	int	Value from the GraphOptions enumeration.
data	void**	Address of a pointer that will be set to point to the specified option value for the graph.
dataLength	unsigned int*	Length in bytes of the buffer pointed to by the data parameter.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.9 mvncSetDeviceOption Function

This function is used to set an option of the device. The available options can be found in the DeviceOptions enumeration.

```
mvncStatus mvncSetDeviceOption(void *deviceHandle, int option,
const void *data, unsigned int datalength);
```

Arguments

Name	Type	Description
deviceHandle	void*	Pointer obtained by calling the mvncOpenDevice function that specifies the NCS device.
option	int	Integer from the DeviceOptions enumeration.
data	const void*	Pointer to the value of the device option to set. Type of data depends on what option is specified.
datalength	unsigned int	Length in bytes of the buffer pointed to by the data parameter.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.10 mvncGetDeviceOption Function

This function is used to get optional information from the device. The available options can be found in the DeviceOptions enumeration.

```
mvncStatus mvncGetDeviceOption(void *deviceHandle, int option,
void **data, unsigned int *datalength);
```

Arguments

Name	Type	Description
deviceHandle	void*	Pointer obtained by calling the mvncOpenDevice function that specifies the NCS device.
option	int	Integer from the DeviceOptions enumeration.
data	void**	Address of a pointer that will be set to point to the specified option value.
datalength	unsigned int*	Returned data length expressed in bytes of the buffer pointed to by the data parameter.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

3.2.11 mvncCloseDevice function

This function is used to cease communication and reset the device.

```
mvncStatus mvncCloseDevice(void *deviceHandle);
```

Arguments

Name	Type	Description
deviceHandle	void*	Pointer obtained from a previous call to mvncOpenDevice function that specifies the NCS device.

Returns

This function returns an appropriate value from the mvncStatus enumeration.

4.0 Python API

4.1 Enumerations

This section describes the following enumeration subclasses: Status, GlobalOption, DeviceOption and GraphOption.

4.1.1 Class Status(Enum)

The Status class is an enumeration that defines the status codes returned from most calls to the C API functions. If the underlying C API returns a non-zero status, an exception is raised with the corresponding status. The possible status codes are shown below.

The Status class is defined as follows:

```
>>> class Status(Enum):
...     OK = 0
...     BUSY = -1
...     ERROR = -2
...     OUT_OF_MEMORY = -3
...     DEVICE_NOT_FOUND = -4
...     INVALID_PARAMETERS = -5
...     TIMEOUT = -6
...     MVCMDNOTFOUND = -7
...     NODATA = -8
...     GONE = -9
...     UNSUPPORTEDGRAPHFILE = -10
...     MYRIADERROR = -11
```

Enumerators

Enumerator Value	Description
MVNC_OK = 0	The function call worked as expected.
MVNC_BUSY = -1	The device is busy, retry later.
MVNC_ERROR = -2	An unexpected error was encountered during the function call.
MVNC_OUT_OF_MEMORY = -3	The host is out of memory.
MVNC_DEVICE_NOT_FOUND = -4	There is no device at the given index or name.
MVNC_INVALID_PARAMETERS = -5	At least one of the given parameters is invalid in the context of the function call.
MVNC_TIMEOUT = -6	Timeout in the communication with the device.

Enumerator Value	Description
MVNC_MVCMDNOTFOUND = -7	The file named *MvNCAPI.mvcmd* is installed in the *mvnc* directory. This message means that the file has been moved or installer failed.
MVNC_NODATA = -8	No data to return.
MVNC_GONE = -9	The graph or device has been closed during the operation.
MVNC_UNSUPPORTEDGRAPHFILE = -10	The graph file is corrupt or may have been created with an incompatible prior version of the NCS toolkit. Try to recompile the graph file with the version of the Toolkit that corresponds to the API version.
MVNC_MYRIADERROR=-11	An error has been reported by the Movidius™ VPU. Use MVNC_DEBUGINFO.

4.1.2 Class GlobalOption(Enum)

The class GlobalOption is an enumeration that defines the options that are used for the SetGlobalOption and the GetGlobalOption functions.

```
>>> class GlobalOption(Enum):
...     LOGLEVEL = 0
...
```

Enumerators

Enumerator value	Description
LOGLEVEL = 0	0=nothing is printed, 1=errors only, 2=verbose.

4.1.3 Class DeviceOption(Enum)

The class DeviceOption is an enumeration that defines the options that are used for the SetDeviceOption and the GetDeviceOption functions.

```
>>>Class DeviceOption(Enum):
...    THERMAL THROTTLING LEVEL = 1002
```

Enumerators

Enum member values	Description
THERMAL_THROTTLING_LEVEL = 1002	Returns 1 if lower guard temperature threshold of chip sensor is reached. This indicates short throttling time is in action between inferences to protect the device. Returns 2 if upper guard temperature of chip sensor is reached. This indicates long throttling time is in action between inferences to protect the device.

4.1.4 Class GraphOption(Enum)

The GraphOption class is an enumeration that defines the options that are used for the SetGraphOption and the GetGraphOption functions.

```
>>>Class GraphOption(Enum):
...    DONTBLOCK = 2
...    TIMETAKEN = 1000
...    DEBUGINFO = 1001
...
```

Enumerators

Enumerator Values	Description
DONTBLOCK = 2	LoadTensor will return BUSY instead of blocking, GetResult will return NODATA instead of blocking.
TIMETAKEN = 1000	Return a NumPy float array [`numpy.array()`] of inference times per layer in float data type.
DEBUGINFO = 1001	Return a string with the error text as returned by the device.

4.2 Global functions

4.2.1 EnumerateDevices Function

This function is used to get a list of the names of the devices present in the system.

```
>>> def EnumerateDevices()
...
```

Parameters

Parameter	Description
None	

Return value

List of device name strings.

4.2.2 SetGlobalOption Function

This function is used to set a global option. The available options can be found in the GlobalOption enumeration in section 4.1.2.

```
>>> def SetGlobalOption(opt, value)
...
```

Parameters

Parameter	Description
opt	The GlobalOption option value that specifies which option to set. See section 4.1.2.
value	The value to which the specified GlobalOption will be set.

Return value

No return value

4.2.3 GetGlobalOption Function

The GetGlobalOption function is used to list the global options. The available options can be found in the GlobalOption enumeration in section 4.1.2.

```
>>> def GetGlobalOption(opt)
...
```

Parameters

Parameter	Description
opt	The GlobalOption value that specifies which option to get. See section 4.1.2.

Return value

The value of the specified GlobalOption.

4.3 Device Class

This section presents the functions that are specific to the Device class.

4.3.1 _init_ Method

The _init_ method is used to initialize a device object:

```
>>> def _init_(name)
...
```

Parameters

Parameter	Description
name	The device name as returned by the EnumerateDevices function. See section 4.2.1.

Return values

No return value

4.3.2 OpenDevice Function

This function is used to initialize the device.

```
>>> def OpenDevice()
...
```

Parameters

Parameter	Description
None	

Return values

No return value

4.3.3 CloseDevice Function

This function is used to cease communication and reset the device.

```
>>> def CloseDevice()
...
```

Parameters

Parameter	Description
None	

Return value

No return value

4.3.4 SetDeviceOption Function

This function is used to set an option for the device; see DeviceOption enumeration.

```
>>> def SetDeviceOption(opt, value)
...
```

Parameters

Parameter	Description
opt	The DeviceOption value that specifies which option to set.
value	The value to which the specified option will be set.

Return value

No return value

4.3.5 GetDeviceOption Function

This function is used to list the option names and associated values for a device.

```
>>> def GetDeviceOption(opt)
...
```

Parameters

Parameter	Description
opt	The DeviceOption value that specifies which option to get.

Returns value

The value of the specified DeviceOption.

4.4 Graph Class

This section presents the functions that are specific to the Graph class.

4.4.1 AllocateGraph Function

This function is used to allocate a graph on the device and create a handle which can be used for other API function calls such as LoadTensor and GetResult.

```
>>> def AllocateGraph(graphFile)
...
```

Parameters

Parameter	Description
graphFile	Binary graph file

Returns

A Graph object to be used to perform operations on the device.

4.4.2 DeallocateGraph Function

This function is used to deallocate a graph on the device.

Note: This is a reserved call and may not be implemented in all versions.

```
>>> def DeallocateGraph()
...
```

Parameters

Parameter	Description
None	

Return value

No return value

4.4.3 SetGraphOption Function

This function is used to set an option for the graph. The available options can be found in the GraphOption enumeration in section 4.1.4.

```
>>> def SetGraphOption(opt, value)
...
```

Parameters

Parameter	Description
opt	The GraphOption value that specifies which option to set. See section 4.1.4.
value	The value to which the specified GraphOption will be set.

Return value

No return value

4.4.4 GetGraphOption Function

This function is used to list the options set for the graph.

```
>>> def GetGraphOption(opt)
...
```

Parameters

Parameter	Description
opt	The GraphOption value that specifies which option to get. See section 4.1.4.

Return values

The value of the specified GraphOption.

4.4.5 LoadTensor Function

This function is used to initiate an inference on this Graph via the associated NCS device.

```
>>> def LoadTensor(inputTensor, userObj)
...
```

Parameters

Parameter	Description
inputTensor	Input data on which an inference will be run. The data must be passed in a NumPy ndarray of half precision floats (float16).
userObj	A user-defined parameter that is returned by the getResult function along with the inference result for this tensor.

Return values

True normally.

False if busy (in case of non-blocking mode; it would block in blocking mode).

4.4.6 GetResult Function

This function is used to retrieve the results. The function blocks if there are no inference results available.

```
>>> def GetResult()
...
```

Parameters

Parameter	Description
None	

Return values

None, None if there is no data and in non-blocking mode (it would block in blocking mode).

Otherwise, a NumPy ndarray of half-precision floats (float16) representing inference results and a user-defined parameter previously passed to LoadTensor.

5.0 Examples

The purpose of this section is to show how to run the examples included with the API.

5.1 C Examples

The three binaries ncs-check, ncs-threadcheck and ncs-fullcheck are in the <path to API>/ncapi/c_examples directory. They can be invoked from within that directory in one of these ways:

```
$ ./ncs-check [-l<loglevel>] -1
$ ./ncs-check [-l<loglevel>] -2
$ ./ncs-check [-l<loglevel>] [-c<count>] <network directory>
$ ./ncs-threadcheck [-l<loglevel>] [-c<count>] <network directory>
$ ./ncs-fullcheck [-l<loglevel> [-c<count>] <network directory> <picture
file>
```

`-l<loglevel>` is an option to enable verbose output. The loglevel value can be 0 (no log output), 1 (errors only) or 2 (verbose output).

`-1` opens one device and then closes it without further actions.

`-2` opens two devices and then closes both without further actions.

`-c<count>` is an option to set the number of inferences to perform (default 2).

`<network directory>` is the directory that contains graph, stat.txt, categories.txt and inputsize.txt.

`<picture file>` is a parameter that provides information about the image size. This parameter is required for the `ncs-fullcheck` command.

The `ncs-check` and `ncs-threadcheck` commands open the device, allocate a graph by sending the graph file present in the given directory, send some random data representing the input, and get the result, `<count>` times. The results are not printed, as they have no sense, but the profiling data is printed.

The `ncs-fullcheck` command requires the `<picture file>` parameter so that the image can be resized to the appropriate size, preprocessed and sent to the NCS. Classification results are printed back. The `<count>` value is 2 by default, because in the first run some data is cached so the times will be lower in the second run.

The `ncs-threadcheck` command does the same thing as the `ncs-check` command, but it uses two threads to show a threaded approach.

5.2 Python Examples

Only Python 3.x is supported; Python 2.x is not supported.

Samples have been tested with Python3.4 on Raspberry PI running Raspian Jessie, and Python 3.5 on Ubuntu 16.04 on x86-64.

age_gender.py and classification_example.py depend on OpenCV and OpenCV python3 bindings being installed.

For Ubuntu:

OpenCV is installed by the Toolkit setup.sh. Run the install_opencv.sh script in the same directory to install python3 bindings.

For Raspberry PI Raspian Jessie:

OpenCV with Python3 bindings must be installed; tested under OpenCV-3.0.0.

USB cameras were tested, no MIPI cameras were tested.

Included samples are as follows; subject to change without notice:

- **age_gender.py** downloads an image of a human and attempts to determine age or gender, depending on command line parameter.
- **classification_example.py** shows how to use various networks to classify an image of a cat, based on command line parameter.
- **ncs_camera.py** and **stream_infer.py** are similar examples showing how to use a USB camera to generate continuous inferences. stream_infer.py is simplified with detailed instructions in stream_infer/readme.*.
    - For Raspberry PI, an OpenGL sink is needed. Use the –opengl command line argument with ncs_camera.py. For stream_infer.py, change SINK_NAME to use glimagesink in the code.
    - For Raspberry Pi, gstreamer-1.0 and python3-gst-1.0 bindings are needed. Install using

      $sudo apt install gstreamer-1.0

      $sudo apt install python3-gst-1.0