Movidius
an Intel company

Movidius™ Neural Compute Stick

Toolkit Documentation

July 2017

Contents

Movidius
an Intel company

Revision History

Date	Revision	Description
July 2017	1	Initial release

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

Terminology

Term	Description
Caffe	A deep learning framework used to develop networks that can be compiled to run on the NCS
CNN	Convolutional neural network; a type of DNN network
DNN	Deep neural network
Inference	The act of comparing input to a network knowledge base, a subject's attributes can be inferred
NCS	Neural compute stick
NCS SDK	A set of software tools and a host API for the NCS
VPU	Visual processing unit

Reference Documents

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

1.0 *Introduction*

The Movidius™ Neural Compute Toolkit and Movidius™ Neural Compute Stick (NCS) enable rapid prototyping, validation and deployment of Deep Neural Networks (DNN).

The Toolkit is a parsing program that intelligently converts existing networks, creating an optimal conversion specifically for the Movidius™ architecture.

1.1 Toolkit features

- *Compilation*: Translation of network weights and structures from Caffe deep learning frameworks into a Movidius™ Neural Compute Stick compatible format.
- *Profiling*: The generation of detailed, per-layer performance statistics of how your network is running on the Movidius™ Neural Compute Stick.
- *Checking*: Verification of classification accuracy by running inferences with the Movidius Neural Compute Stick.

1.2 The Movidius™ Neural Compute Stick and your network

The Movidius™ Neural Compute Stick (NCS) has twelve vector VLIW ("SHAVE") processors which compute most of the neural network load. The number of active SHAVEs is configurable in the toolkit to 1,2,4,8 or 12.

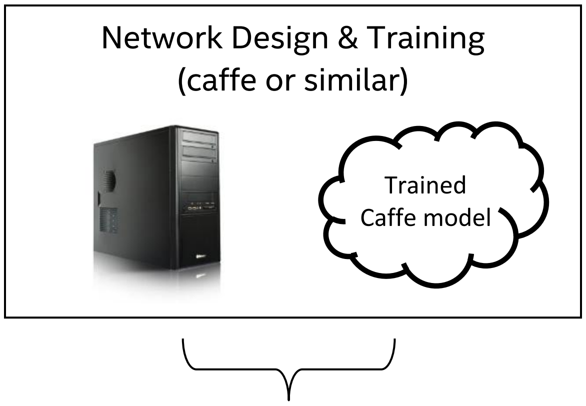
The NCS runs neural network stages with half-precision 16-bit floating point.

1.3 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.

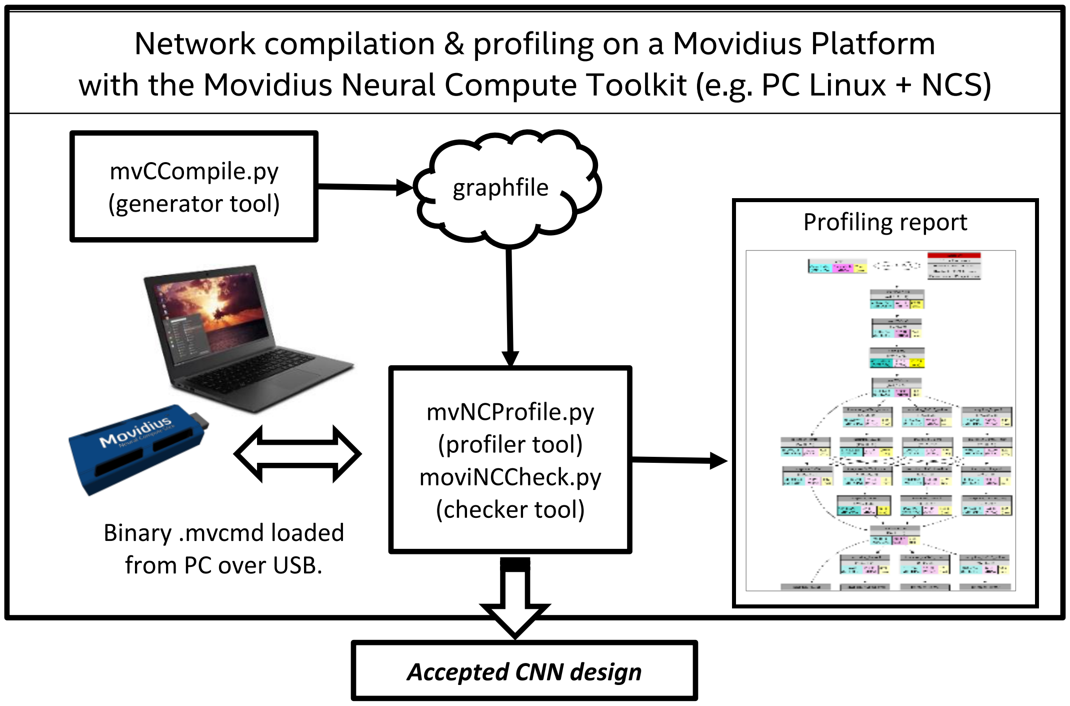
1.3.1 Neural Network development stage (off NCS device)

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



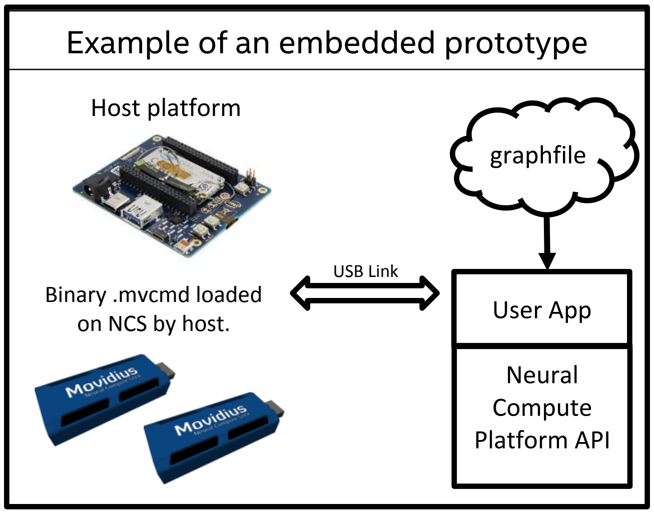
1.3.2 Network compilation & profiling with the Movidius NCS Toolkit

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



1.3.3 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.


Example of an embedded prototype
Host platform
graphfile
USB Link
Binary .mvcmd loaded on NCS by host.
User App
Neural Compute Platform API

2.0 Setup and Installation

This section covers the download of files and documentation to facilitate the installation of the Toolkit and the operation of the NCS device.

2.1 Requirements

The following are the software and hardware requirements to run the Toolkit with the neural network of your choice.

2.1.1 Host computer

Ubuntu Linux* 16.04 LTS x86-64-bit with a USB 2.0 port and an active internet connection.

Note: All steps listed in this document are based on a fresh install of the operating system.

2.1.2 Hardware

Movidius™ Neural Compute Stick

2.2 Checking your host computer setup

2.2.1 Ubuntu distribution and version

Issue the following command to verify that your distribution is 64-bit (x86_64):

```
$ uname -m
```

Check that your version of Ubuntu is 16.04 LTS with the following command:

```
$ lsb_release -a
```

2.3 Toolkit installation steps

2.3.1 Toolkit dependencies

The installation script has been tested on clean installs of Ubuntu Linux* 16.04 LTS where Caffe was not already installed.

2.3.2 Step 1 – Download the Movidius™ Neural Compute SDK

Download Movidius™ Neural Compute SDK archive from the *developer.movidius.com* download area.

Create a directory for the SDK.

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

Move the downloaded 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
```

Unpack the Toolkit archive.

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

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

```
$ cd bin
```

2.3.3 Step 2 – Run the automated setup script

The automated install script will attempt to install the packages that it finds missing from your system. It is the user's responsibility to decide whether this may cause any issues with existing applications.

Move to the root of the toolkit release folder, and run the auto-install script:

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

The script asks for the sudo password and suggests a default location for the dependencies and third-party software like Caffe. It is highly recommended that the default locations be used so that other scripts can find needed files.

Note: The installation takes some time to complete as many packages are downloaded and installed; do not cancel the installation before it is complete.

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
```

To confirm the PYTHONPATH is set correctly use the following command and look for similar output:

```
$ echo $PYTHONPATH
:/opt/Movidius/caffe/python:
```

2.3.4 Step 3 – Verify the installation

Issue the following command to verify your installation:

```
$ make check
```

Note: You can also review the installation log file to make sure that all the requirements are met and that there are no remaining conflicts. The log file contains the full terminal output from the setup.sh script.

The log file is of the format *"setup_YY_MM_DD_HH_mm.log"*

2.3.5 Step 4 – Download existing models

Run the *dlnets.sh* script from the data directory to download a selection of existing network models.

```
$ cd data
$ ./dlnets.sh
```

2.3.6 Step 5 - Test your installation of the Toolkit

See the Demo projects section for ways to test your Toolkit installation.

2.4 Troubleshooting the installation

2.4.1 PYTHONPATH

The PYTHONPATH points to the version of Caffe software that was installed by the setup script. You can use the `export PYTHONPATH` command to change this.

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
```

2.4.2 Support forum

Additional support and information can be found on the ncsforum.movidius.com site.

3.0 *Using the Toolkit*

The Toolkit provides tools to enable rapid profiling, validation and tuning of CNNs. The Toolkit also includes a tool to compile CNNs to binary graph files that the NCS can load and execute from within a user's software application.

3.1 Toolkit utilities

The toolkit is composed of three utilities:

- *mvNCCompile.pyc* (the compiler)
- *mvNCProfile.pyc* (the profiler)
- *mvNCCheck.pyc* (the checker)

3.2 Toolkit directory structure overview

Directory	Purpose
<path-to-SDK>\bin	The *bin* directory is the **root** directory to for the Toolkit. Most utilities should be executed from within this directory. The bin directory contains: • The Toolkit utilities • Example Makefile • Various other toolkit files and libraries
<path-to-SDK>\Controllers	Contains internal Python code for Toolkit Utilities
<path-to-SDK>\Models	Contains internal Python code for Toolkit Utilities
<path-to-SDK>\mvnc	Contains internal Python code for Toolkit Utilities
<path-to-SDK>\Views	Contains internal Python code for Toolkit Utilities
<path-to-SDK>\data	Contains example Caffe models and .prototxt files. By default only the .prototxt files are in this directory, but the dlnets.sh script file is provided to download the Caffe models and generate NumPy mean files for the example models.
<path-to-SDK>\examples	Contains the top5_over_a_dataset.py example Python program and a dataset.
<path-to-SDK>\ncapi	API directory

Movidius
an Intel company

3.3 Compiler

The compiler is used to create a graph container. This is an optimized binary file that can be processed by the NCS.

3.3.1.1 Usage

The command to use the compiler has the following format:

```
$ python3 ./mvNCCompile.pyc <network.prototxt> [...]
```

Argument	Required?	Description
`<network.prototxt>`	Required	Name of the prototxt (Caffe) file of the network. If a corresponding Caffe model file is found, it will be used for the weights, unless -w is provided too.
`-w <weights file>`	Optional	Weights file.
`-s <MAX number of shaves>`	Optional	Default: 1 Selects the maximum number of SHAVEs (1,2,4,8 or 12.) to use for network layers. Note: The NCS runtime code may use less than the MAX SHAVE value for some layers where measurements have typically shown no inference performance degradation (and consequently a power benefit) of using fewer SHAVEs.
`-in <input node name>`	Optional	By default the network is processed from the input tensor. This option allows a user to select an alternative start point in the network. This enables partial network processing. When used together with the -on option a user can isolate one or more layers in a network for analysis.
`-on <output node name>`	Optional	By default the network is processed through to the output tensor. This option allows a user to select an alternative end point in the network. This enables partial network processing. When used together with the -in option a user can isolate one or more layers in a network for analysis.
`-is <in. width> <in. height>`	Optional	For networks that do not have dimension constraints on the input tensor, this option can be used to set the desired input dimensions.

		Only two dimensions are defined because the batch size is always 1 and the number of color planes is assumed to be 3.
`-o <path>`	Optional	Output graph container filename. If not provided, “graph” will be used.

3.3.1.2 Example

The following is an example of a Python command that is issued from the *bin* directory, where *caffemodels/<MyModel>.prototxt* is an example of network file location to be replaced with the path to the network file of interest for your application:

```
        $ python3 ./mvNCCompile.pyc caffemodels/<MyModel>.prototxt
-w caffemodels/<MyModel>.caffemodel -o caffemodels/<MyModel>
```

3.3.2 Checker

The checker runs a single inference on the NCS, allowing for the calculation of classification correctness. It prints the top-5 classification indexes and their probabilities for both the NCS outputs (Result) and Caffe outputs (Expected). It also prints the numerical deviation between the two outputs using several metrics.

3.3.2.1 Usage

The command to use the checker has the following format:

```
        $ python3 ./mvNCCheck.pyc <network.prototxt> [...]
```

Argument	Required?	Description
`<network.prototxt>`	Required	Name of the prototxt (Caffe) file of the network. If a corresponding Caffe model file is found, it will be used for the weights, unless -w is provided too.
`-w <weights file>`	Optional	Weights file.
`-s <number of shaves>`	Optional	Default: 1 Description: Selects the maximum number of SHAVEs to use for network layers. Note: The NCS runtime code may use less than the MAX SHAVE value for some layers where measurements have typically shown no inference performance degradation (and consequently a power benefit) of using fewer SHAVEs

`-in <input node name>`	Optional	By default the network is processed from the input tensor. This option allows a user to select an alternative start point in the network. This enables partial network processing. When used together with the -on option a user can isolate one or more layers in a network for analysis.
`-on <output node name>`	Optional	By default the network is processed through to the output tensor. This option allows a user to select an alternative end point in the network. This enables partial network processing. When used together with the -in option a user can isolate one or more layers in a network for analysis.
`-is <in. width> <in. height>`	Optional	For networks that do not have dimension constraints on the input tensor, this option can be used to set the desired input dimensions. Only two dimensions are defined because the batch size is always 1 and the number of color planes is assumed to be 3.
`-i <image>`	Optional	Image to use as input to validation run. If not set, a randomly generated image will be used.
`-id <number>`	Optional	Expected id for Top-1 validation.
`-S <number>`	Optional	Scale each value of the input by this amount. E.g. if the network expects input values in the range 0-255, put 255 here (1 is default, as the range 0-1 is the default).
`-M <number or numpy file>`	Optional	Subtract this from the input (applied after scale). E.g. If the network expects a mean file to be subtracted from the input image, put it here.

3.3.2.2 Example

The following is an example of a command that uses the checker:

```
$ python3 ./mvNCCheck.pyc caffemodels/../<MyModel>.prototxt -w
caffemodels/<MyModel>.caffemodel -i image.jpg -S 255 -s 12 -M
./data/mean.npy
```

3.3.3 Profiler

Use the profiler to prototype your network to best fit Movidius™ proprietary architecture. A single on-chip inference generates enough information to provide a detailed stage-by-stage breakdown of where the bottlenecks are in your system. The profiler returns a console output as well as an HTML report.

3.3.3.1 Usage

The command to use the profiler has the following format:

```
$ python3 ./mvNCProfile.pyc <network.prototxt> [...]
```

Argument	Required?	Description
`<network.prototxt>`	Required	Name of the prototxt (Caffe) file of the network. If a corresponding Caffe model file is found, it will be used for the weights, unless -w is provided too.
`-w <weights file>`	Optional	Weights file.
`-s <MAX number of shaves>`	Optional	Default: 1 Description: Selects the maximum number of SHAVEs to use for network layers. Note: The NCS runtime code may use less than the MAX SHAVE value for some layers where measurements have typically shown no inference performance degradation (and consequently a power benefit) of using fewer SHAVEs
`-in <input node name>`	Optional	By default the network is processed from the input tensor. This option allows a user to select an alternative start point in the network. This enables partial network processing. When used together with the -on option a user can isolate one or more layers in a network for analysis.
`-on <output node name>`	Optional	By default the network is processed through to the output tensor. This option allows a user to select an alternative end point in the network. This enables partial network processing. When used together with the -in option a user can isolate one or more layers in a network for analysis.
`-is <in. width> <in. height>`	Optional	For networks that do not have dimension constraints on the input tensor, this option can be used to set the desired input dimensions. Only two dimensions are defined because the batch size is always 1 and the number of color planes is assumed to be 3.

The *Makefile* included in the *<path-to-toolkit>/bin* folder is provided to showcase some scenarios.

Issue the following command to view the list of available targets:

```
$ make help
```

3.4 Examples

The networks downloaded during the installation step downloading models can be processed with the Toolkit for use on the NCS. While the networks downloaded by the script may change without notice, the base set should include the following:

- AlexNet
- GoogLeNet
- SqueezeNet
- Age
- Gender
- LeNet8

3.4.1 Make Examples

Run the `make help` command to view the list of included examples. Make *example00* to get started with the compiler. Issue the following command to build and execute the example:

```
$ make example00
```

3.4.2 Test Script

The following is a command example for a test script. This command must be issued from the *bin* directory:

```
$ python3 ./examples/top5_over_a_dataset.py
examples/data/test/ data/lenet8.prototxt
data/lenet8.caffemodel
```

You can use the source code as starting point for your development.