Hands-on session

Prerequisites:

1. A laptop with at least two USB ports (although some USB 2 hubs will be available)
2. A laptop loaded with VMware® Workstation 14(+) Pro
3. At least 32GB of disk space free
4. A USB stick loaded with a VMware image
5. One Movidius Neural Compute Stick (two maybe needed to be shared)
6. Validate system environment as described in below section
7. Internet connection
8. Commands are prefixed by a ‘$’ and next-line code continuation is marked by ‘\’

Validating the System Setup

After starting the VMware workstation load the VMware image provide and start the image.

Default user:
Default password is:

Check your environment

First test for the correct version of ubuntu

```
$ uname –a
Linux ubuntu 4.13.0-41-generic #46~16.04.1-Ubuntu SMP Thu
May 3 10:06:43 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

Next test for python3 (also aliased to python) and the *tensorflow* environment. The command should not throw any errors, but a warning about float types. It also tests for the *tensorflow* environment.

```
$ python3 -c ‘import tensorflow as tf’
```

This command should not throw any errors, but a warning about float types. It tests for the tensorflow environment.

The next command tests for the presence of the *ncsdk* toolkit

```
$ which mvNCCompile mvNCCheck mvNCProfile
/usr/local/bin/mvNCCompile
/usr/local/bin/mvNCCheck
/usr/local/bin/mvNCProfile

$ python3 -c 'import mvnc'
```

This command execution should not throw any errors.

```
$ source ./setupvars.sh
[setupvars.sh] OpenVINO environment initialized
```

Note: `python` has been aliased to `python3` so their usage is interchangeable.

End-to End tensorflow CNN MNIST example

Objective:

To create an end-to-end convolutional neural network (CNN) in *TensorFlow* that will run on the host CPU system. It will have several macro parameters that can be manually adjusted in the files to change the accuracy and the final image to be analyzed. The macro parameters will be indicated in the activities. The files are a mixture of bash and python3 files, with the graphics at the end using the matplotlib module. This activity should produce a baseline for comparison to other systems e.g. CPU, Movidius MYRIAD 2, and OpenVino supported systems.

All material for this set of exercises are in `~/tensorflow_MNIST`, in which you should have:

```
$ cd ~/tensorflow_MNIST
$ ls
mnist_input_data/
test_data/
tensorflow_CPU/
tensorflow_Movidius/
tensorflow_Openvino/
```

`mnist_input_data/`

Contains the downloaded MNIST training and testing files.

`test_data/`

Contains: `'*.png'` image files and a `'labels.txt'` and the labels file.

Warning: only use the `'*xxB.png'` files: to simplify the CNN these files have already been processed to the appropriate format.

Part 1. CPU based *tensorflow* CNN

This example creates an end-to-end MNIST CNN in t*ensorflow* that will run on the host system.

You will need to move to the directory *`tensorflow_CPU`*

```
$ cd ~/tensorflow_MNIST/tensorflow_CPU
```

where you'll find the following:

```
clean.sh

Step_01_train_net.py

Step_02_addIO.py

Step_03_graphLoad.py
```

`clean.sh`

Destroys the intermediate files and directories created, including the CNN itself, except for the downloaded MNIST training and testing data files.

To run type:

```
$./clean.sh
```

`Step_01_train_net.py`

Downloads the training and testing datasets if necessary. It then creates, trains and tests a CNN putting the results into

```
Result_01_trained_net/
```

This directory can be deleted using the `clean.sh as mentioned above.` The script is executed using

```
$ python3 Step_01_train_net.py
```

`Step_02_addIO.py`

Adds the appropriately formed and named '*input'* and '*output'* layers and removes the dropout layers, it stores the results in:

```
Result_02_IOAdded/
```

This directory can also be deleted using the `clean.sh.` The script is executed by entering:

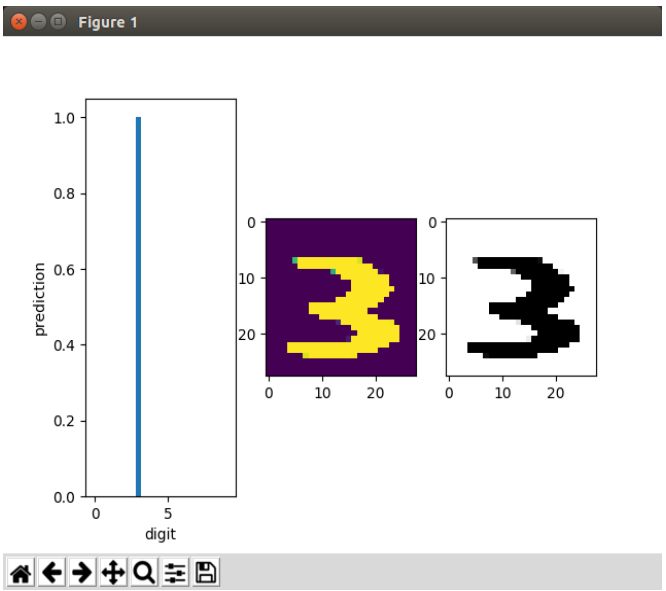
```
$ python3 Step_02_addIO.py
```

`Step_03_graphLoad.py`

Loads and runs the CNN stored in `Result_02_IOAdded/`. The script is executed by entering:

```
$ python3 Step_03_graphLoad.py
```

The output results should look like:



for the input file `'3xxB.png'`, indicating the probability of the digit's value and images of the input and any processing of that image.

Activity 1

We will be training the MNIST CNN on the CPU for **inferencing on the CPU**. Run the python codes in `~/tensorflow_MNIST/tensorflow_CPU` as illustrated:

```
$ python3 Step_01_train_net.py
$ python3 Step_02_addIO.py
$ python3 Step_03_graphLoad.py
```

Activity 2

Adjust the number of iterations (epochs) that the training program runs for and check the accuracy. The parameter to change is:

```
NITER = 200
```

You might want to run multiple examples and plot the accuracy as a function of iterations.

Activity 3

Use a variety of input files for the last stage – look in `~/tensorflow_MNIST/test_data` for suitable `'*xxB.png'` files.

You will need to edit the file `'Step_03_graphLoad.py'` and change the setting for

```
IMAGE = '3xxB.png'
```

On line 11.

Part 2. Introduction to Intel® OpenVINO toolkit

Here we will build example end-to-end MNIST CNNs using tensorflow and OpenVino for deployment on the CPU and Movidius, followed be a demo of existing public deep-learning models.

Part 2.1. Building a CNN with tensorflow for deployment on the CPU.

We will need to move into the directory `~/tensorflow_MNIST/tensorflow_Openvino/CPU`

```
$ cd  ~/tensorflow_MNIST/tensorflow_Openvino/CPU
```

Here you will find four new scripts:

```
classification_sample.py

Step_03_freeze.py

Step_04_ir.sh

Step_05_openVinoRun.sh
```

`Step_03_freeze.py`

This script freezes the tensorflow graph (the tensorflow network definition and the weights) to produce a `'.pb'` file in the directory:

```
Result_03_frozen/
```

`Step_04_ir.sh`

This script produces the `'.xml'` and the `'.bin'` files necessary for the CPU deployment. The script contains the following:

```
mo_tf.py --input_model ./Result_03_frozen/frozen.pb \

         --output_dir ./Result_04_ir
```

Which has a default FP32 representation for the floats.

`Step_05_openVinoRun.sh`

Is a wrapper script to `'classification_sample.py'` script which is a modified classifier that takes into account the IO of the current CNN. It contains the following:

```
python3 classification_sample.py  -m ./Result_04_ir/frozen.xml
```

Activity 1

Run the scripts through their sequence to deploy a CNN with Openvino on the local CPU architecture.

Activity 2

Repeat the activities from the section Part 1.

Part 2.2. Building a CNN with *tensorflow* for deployment with Openvino on the MYRIAD 2.

For this you will need to use the USB 2 hub and plug in at least one Movidius stick.

We will need to move into the directory `~/tensorflow_MNIST/tensorflow_Openvino/MYRIAD`

```
$ cd  ~/tensorflow_MNIST/tensorflow_Openvino/MYRIAD
```

Here you will find the same scripts as in part 1 but scripts:

`Step_04_ir.sh`

`Step_05_openVinoRun.sh`

Have been modified to reflect that the resultant CNN is to be deployed on the Movidius sticks

`Step_04_ir.sh`

This script produces the `'.xml'` and the `'.bin'` files necessary for the MYRIAD deployment. The script contains the following:

```
mo_tf.py --input_model ./Result_03_frozen/frozen.pb   \
         --output_dir ./Result_04_ir --data_type FP16
```

Which sets the floating point representation to FP16 representation for the floats using the `' –data_type'` flag.

`Step_05_openVinoRun.sh`

Is a wrapper script to `'classification_sample.py'` script which is a modified classifier that takes into account the IO of the current CNN. It contains the following:

```
python3 classification_sample.py  -m ./Result_04_ir/frozen.xml
-d MYRIAD
```

where the `'-d MYRIAD'` tells the system that the model is to be deployed on the Movidius stick.

Activity 1

Run the scripts through their sequence to deploy the MNIST CNN with Openvino on the external Movidius stick.

Activity 2

Compare scripts ‘04’ and ’05’ from Part 2.1 and Part 2.2 i.e. the difference in creating suitable models to run on two distinct architectures of the local CPU and the Movidius MYRIAD 2 stick.

Activity 3

Repeat the activities from the previous section: you will need to modify `‘classification_sample.py’for` the input image and number of epochs to run the simulation.

Activity 4

Go to the model demo directory

```
$ cd /opt/intel/computer_vision_sdk/deployment_tools/demo
```

And execute:

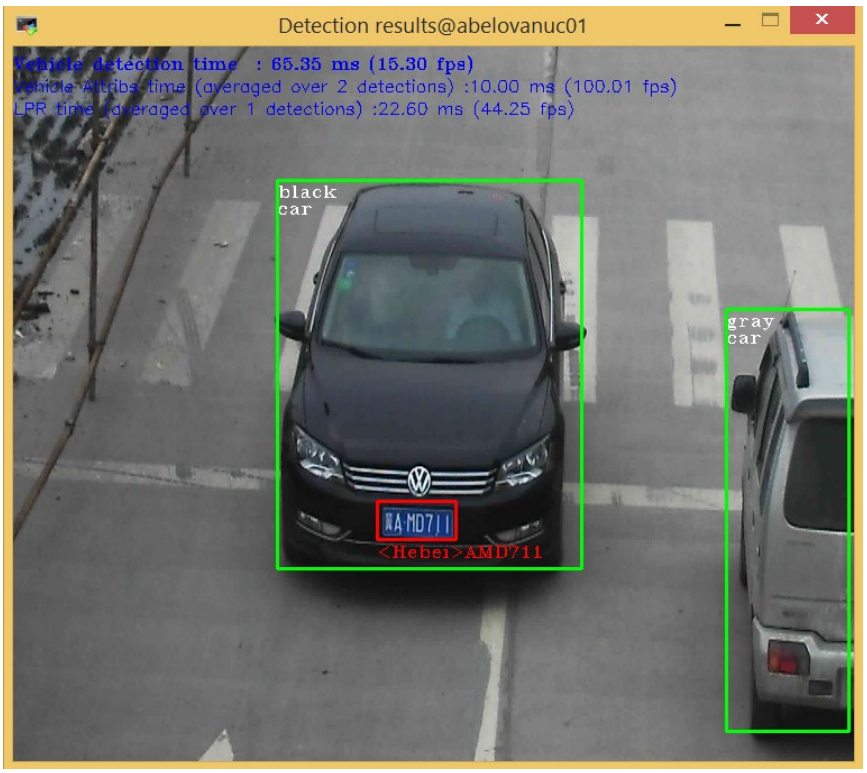
```
$ sudo -E ./demo_security_barrier_camera.sh
```

The expected output should be:

```
InferenceEngine:
        API version ............ 1.0
        Build .................. 9911
[ INFO ] Parsing input parameters
[ INFO ] No extensions provided
[ INFO ] Reading input
[ INFO ] Loading plugin CPU

        API version ............ 1.0
        Build .................. lnx_20180314
        Description ....... MKLDNNPlugin

[ INFO ] Loading network files for VehicleDetection
[ INFO ] Batch size is forced to  1
[ INFO ] Checking Vehicle Detection inputs
[ INFO ] Checking Vehicle Detection outputs
[ INFO ] Loading Vehicle Detection model to the CPU plugin
[ INFO ] Loading network files for VehicleAttribs
[ INFO ] Batch size is forced to 1 for Vehicle Attribs
[ INFO ] Checking VehicleAttribs inputs
[ INFO ] Checking Vehicle Attribs outputs
[ INFO ] Loading Vehicle Attribs model to the CPU plugin
[ INFO ] Loading network files for Licence Plate Recognition (LPR)
[ INFO ] Batch size is forced to  1 for LPR Network
[ INFO ] Checking LPR Network inputs
[ INFO ] Checking LPR Network outputs
[ INFO ] Loading LPR model to the CPU plugin
[ INFO ] Start inference
[ INFO ] Execution successful
```



Activity 5

Go to the model downloader tool directory

```
$ cd /opt/intel/computer_vision_sdk/deployment_tools/demo
```

And execute:

```
$ sudo -E ./demo_squeezenet_download_convert_run.sh
```

Note additional networks may be downloaded

Part 3. Movidius based *tensorflow* CNN

Building a CNN with *tensorflow* for deployment with ncsdk on the Movidius stick, the valdation of the environment was carried out on page 1. But you will need to use the USB 2 hub and plug in at least one Movidius stick.

We will need to move into the directory `~/tensorflow_MNIST/tensorflow_Movidius`

```
$ cd  ~/tensorflow_MNIST/tensorflow_Movidius
```

Here you will find the same scripts as in part 1 for steps 01 and 02, but with different scripts for the third and fourth parts:

```
Step_03_createGraph.sh

Step_04_graphLoad.py
```

Steps 03 and 04 have been modified to reflect that the resultant CNN is to be deployed on the Movidius sticks.

`Step_03_createGraph.sh`

This script creates uses the ncsdk `mvNCCompile` command to create a graph file contains the network layout and the weights for deployment onto the Movidius stick. It contains the following:

```
cd Result_03_movidiusGraph

cp ../Result_02_IOAdded/* .

mvNCCompile IOAdded.meta -s 12 -in input -on output -o IOAdded.graph
```

which identifies the input and out put node layers, and tells the Movious to utilize 12 of the shave processors.

`Step_04_graphLoad.py`

Deploys the graph onto the Movidius neural compute sticks.

Activity 1

We will be training the MNIST CNN on the CPU for **inferencing on the Movidius**. Run the python codes in `~/tensorflow_MNIST/tensorflow_Movidius` as illustrated:

```
$ python3 Step_01_train_net.py

$ python3 Step_02_addIO.py

$ Step_03_createGraph.sh

$ Step_04_graphLoad.py
```

Activity 2

Adjust the number of iterations (epochs) that the training program runs for and check the accuracy. The parameter to change is:

```
NITER = 200
```

You might want to run multiple examples and plot the accuracy as a function of iterations.

Activity 3

Use a variety of input files for the last stage – look in `~/tensorflow_MNIST/test_data` for suitable `'*xxB.png'` files.

You will need to edit the file `'Step_04_graphLoad.py'` and change the setting:

```
IMAGE = '3xxB.png'
```

Activity 4

Edit the number of shave processors used in the file `Step_03_createGraph.sh` set with the `-s` flag

```
mvNCCompile IOAdded.meta -s 12 -in input -on output \

                         -o IOAdded.graph
```

Experiment with the values from **-s 12** to **-s 1,2,4,8** and compare the wall time and inference

Activity 5

Use **mvNCProfile** to generate a graph report:

```
$ mvNCProfile -s 12 ./Result_03_movidiusGraph/IOAdded.meta \
              -in input \
              -on output
```

Use a web browser like *firefox* to view the resultant ‘./output.gv.svg’ or ‘./output_report.html’ files.

Activity 6

Use **mvNCCheck** to compare the results between the host and the Movidius stick

```
mvNCCheck ./Result_03_movidiusGraph/IOAdded.meta \
          -in input \
          -on output
```

Where in and on refer to the input and output node layers, similarly the other layers can be referenced using the layer names outlined in Activity 5.

For some of these examples you will need two Movidius sticks, a USB 2.0 hub, and either an internal or external (preferable!) USB camera.

Activity 7 – inception_v4

Navigate to the following directory

```
$ cd ~/ncappzoo/tensorflow/inception_v4
```

In it you will find a `Makefile`, which will take care of the preparations for this example: download data files, graph files, category files etc. In order to run this application, you will need to issue the *make* command.

```
$ make
```

To run the example type:

```
$ python run.py
```

The line 15, in run.py file, has a reference to an image stored in

`image_filename = path_to_images + 'nps_electric_guitar.png'`.

By passing the image through the inference engine the program will attempt to determine the image content and list probable outcomes.

From the `path_to_images` variable, point the `image_filename` at another valid image to try and determine its contents.

Activity 8 – tiny_yolo

Navigate to the directory

```
$ cd ~/ncappzoo/apps/stream_ty_gn
```

Run

```
$ python stream_ty_gn.py
```

What do you get?

Activity 9 – gender_age

Navigate to the directory

```
$ cd ~/ncappzoo/apps/gender_age_lbp/cpp
```

And run:

```
$ ./gender_age_lbp
```

How old and what gender are you?

(Note there is an assertion that can make this program unstable!)