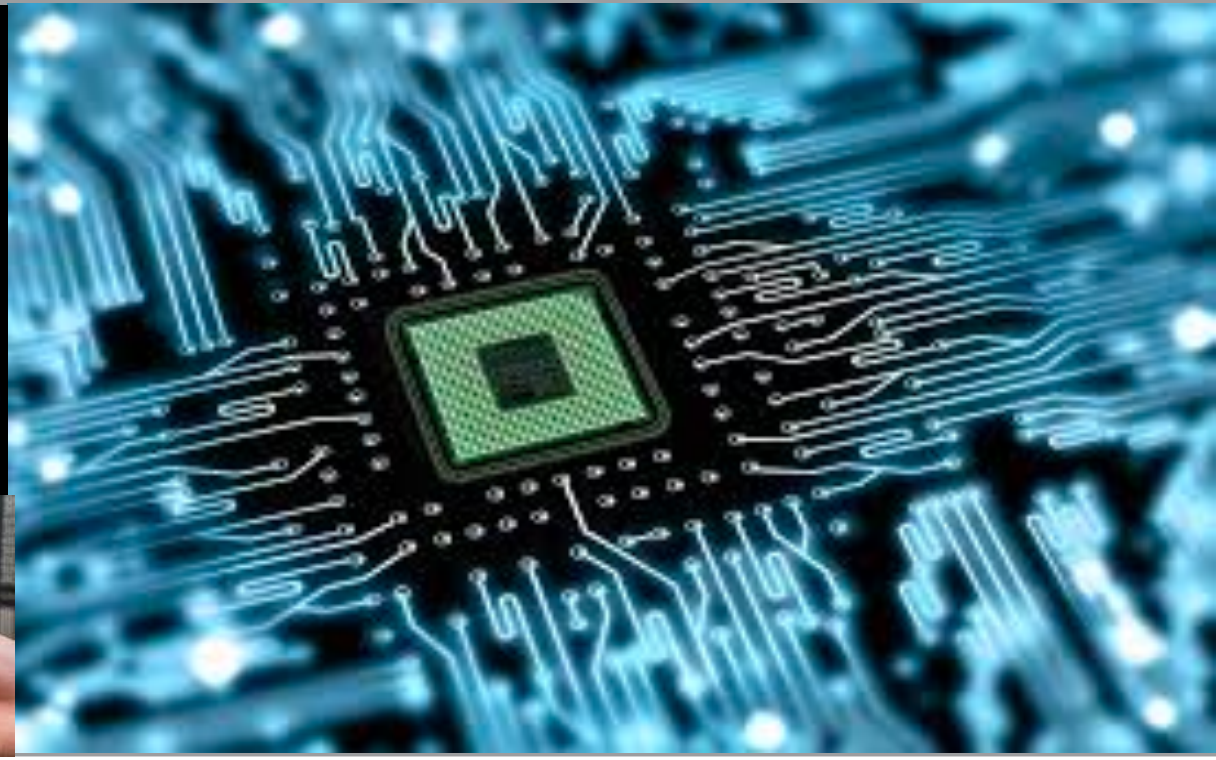


IEEE

Crash Course in Tensorflow and NCS

NUI Galway
OÉ Gaillimh

Center for Cognitive, Connected and Computational Imaging
College of Engineering, Science & Informatics, NUI Galway

Cognitive, Connected & Computational Imaging

A crash course in Tensorflow

Tensorflow is HUGE!

We cover only the small subset of tensorflow that works well with the Movidus NCS.
Online you will find this referred to as “tensorflow-core” or “low level” tensorflow

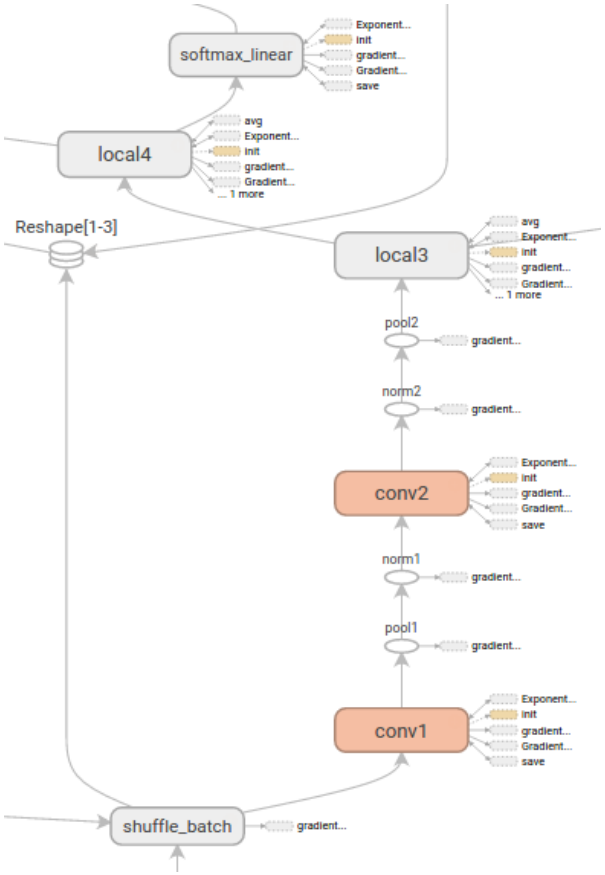
You can use Tensorflow in python with:
import tensorflow as tf

A crash course in Tensorflow

Using Tensorflow involves two steps:

- Building the graph
- Running the graph.

These steps are invoked separately.



A crash course in Tensorflow – Building a graph

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
```

In this example, neither a nor b has a value yet.
All we've done is set up a computation graph!

A crash course in Tensorflow – Running a graph

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
# Nothing has executed yet. Neither a nor b has a
value yet.
sess = tf.Session() # invoke session
print(sess.run(total)) # run graph in session.
# Our addition has been executed. The above line
will print 7
```

A crash course in Tensorflow – Feeding a dictionary with variables.

```
# Define graph using placeholders
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y
# Execute graph, specifying values.
print(sess.run(z, feed_dict={x: 3, y: 4.5}))
# output: 7.5
print(sess.run(z, feed_dict={x: [1, 3], y: [2, 4]}))
# output: [ 3.  7.]
```

A crash course in Tensorflow – Training a simple network

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
print(sess.run(y_pred))
```

Wait a second, what does all that mean? (line by line)

In machine learning X is usually the thing we want to make predictions on.
It could be an image or a sound or any other type of data.
This is also often called the “input” to the neural network.
The code:

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
```

is just setting up the a constant called x that we want to use for input later.

Wait a second, what does all that mean? (line by line)

In machine learning Y is often called the “target” or “labels”.

In an image classification task it would be the “class” of an image. In a pedestrian detector y would be a bounding box around a pedestrian.

Y is also the output of the machine learning model.

Wait a second, what does all that mean? (line by line)

The code:

```
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
```

Sets up a constant called y_true which contains the "ground truth" or true labels the network uses to learn from.

In a real example, these would not be constants but instead would be loaded from a database. In such a case we would use tf.placeholder instead.

A crash course in Tensorflow – Training a simple network

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
print(sess.run(y_pred))
```

Wait a second, what does all that mean? (line by line)

```
linear_model = tf.layers.Dense(units=1)
```

Initializes a very simple layer containing just one unit. A dense layer is sometimes called a “fully connected” layer.

The default activation function for a Dense layer is linear (y=x), but any activation function could be passed as an argument if desired. For example, relu would be max(0,x).

Nonlinear activation functions are often called nonlinearities.

Wait a second, what does all that mean? (line by line)

y_pred = linear_model(x)

y_pred is used to store “predictions” from the model.

This line also allows tensorflow to understand how we want to build the neural network. This is the simplest possible neural network containing only one hidden layer with only one unit.

Deep neural networks stack many of these together to model complicated relationships but even very simple models like this can still be powerful.

A crash course in Tensorflow – Training a simple network

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
print(sess.run(y_pred))
```

Wait a second, what does all that mean? (line by line)

```
loss = tf.losses.mean_squared_error(labels=y_true,
predictions=y_pred)
```

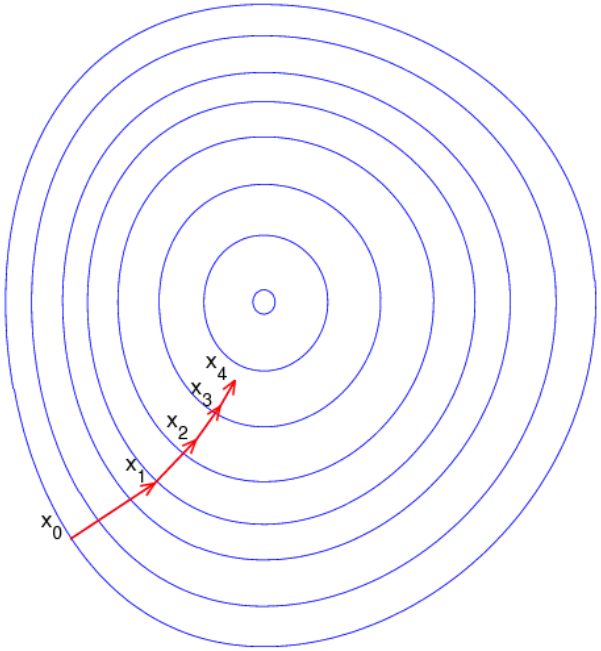
In neural networks, a loss variable works to tell the optimizer how close the output from the network (y_pred) is to the correct value (y_pred). In this case we use the mean squared error but other losses are also common.

The loss variable can be calculated from just one sample, from the entire dataset, or more commonly from a mini-batch of samples.

Wait a second, what does all that mean? (line by line)

```
optimizer =
tf.train.GradientDescentOptimizer(0.01)
```

This creates a gradient descent optimizer. Gradient descent is used to find the “minimum” of a function. It uses a learning rate of 0.01.



In deep learning finding a “local minima” is the goal. This is the place that is close to the solution.

A crash course in Tensorflow – Training a simple network

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
print(sess.run(y_pred))
```

Wait a second, what does all that mean? (line by line)

```
train = optimizer.minimize(loss)
```

This line tells tensorflow that it should use the gradient descent optimizer defined previously to minimize the MSE loss function we defined before.

It also tells tensorflow to put this operation on its computation graph in an operation called “train”.

A crash course in Tensorflow – Training a simple network

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
print(sess.run(y_pred))
```

Wait a second, what does all that mean? (line by line)

```
init = tf.global_variables_initializer()
```

The above line creates a tensorflow variable initializer, which when run in the session defined below will build the graph, initialize values and perform operations.

```
sess = tf.Session()
sess.run(init)
```

The above lines create a session handle for us to run our code in.

A crash course in Tensorflow – Training a simple network

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
print(sess.run(y_pred))
```

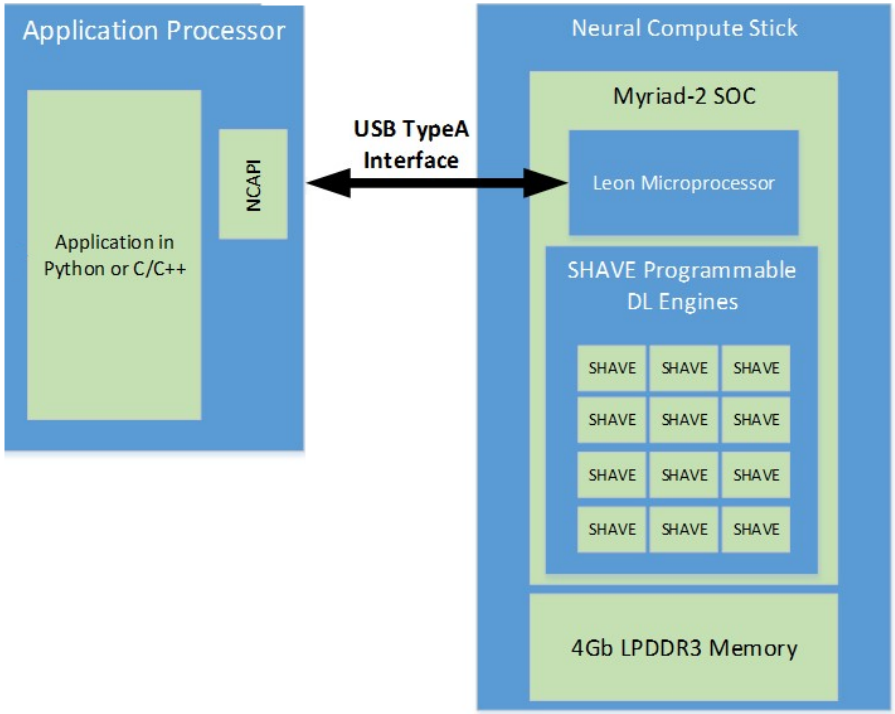
Wait a second, what does all that mean? (line by line)

```
for i in range(100):
  _, loss_value = sess.run((train, loss))
```

This puts everything we've covered before together. We "execute our graph" 100 times and train a very simple model that learns to create predictions using all the previously defined operations.
This is where all the training happens.

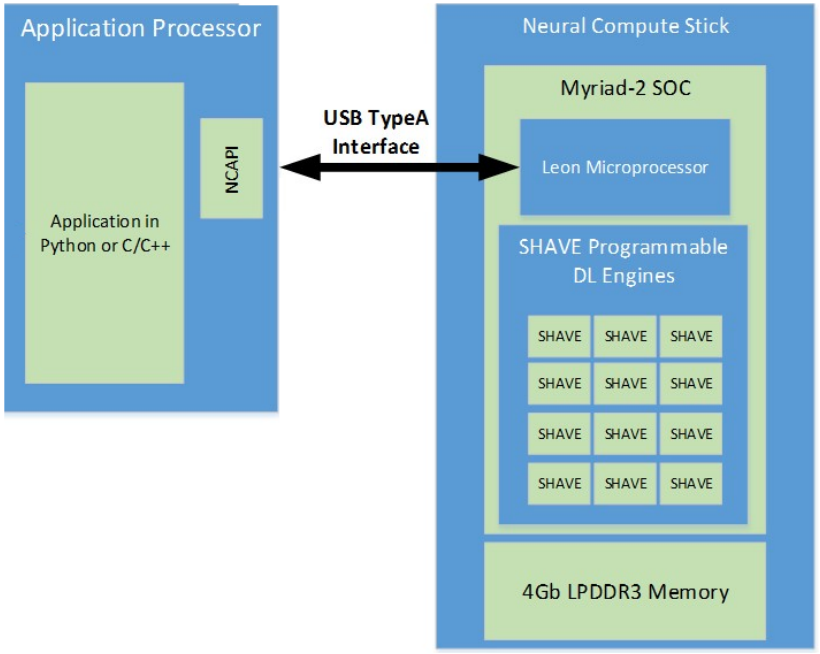
Introducing the Movidius Neural Compute Stick

The NCS is able to perform accelerated deep learning inference.

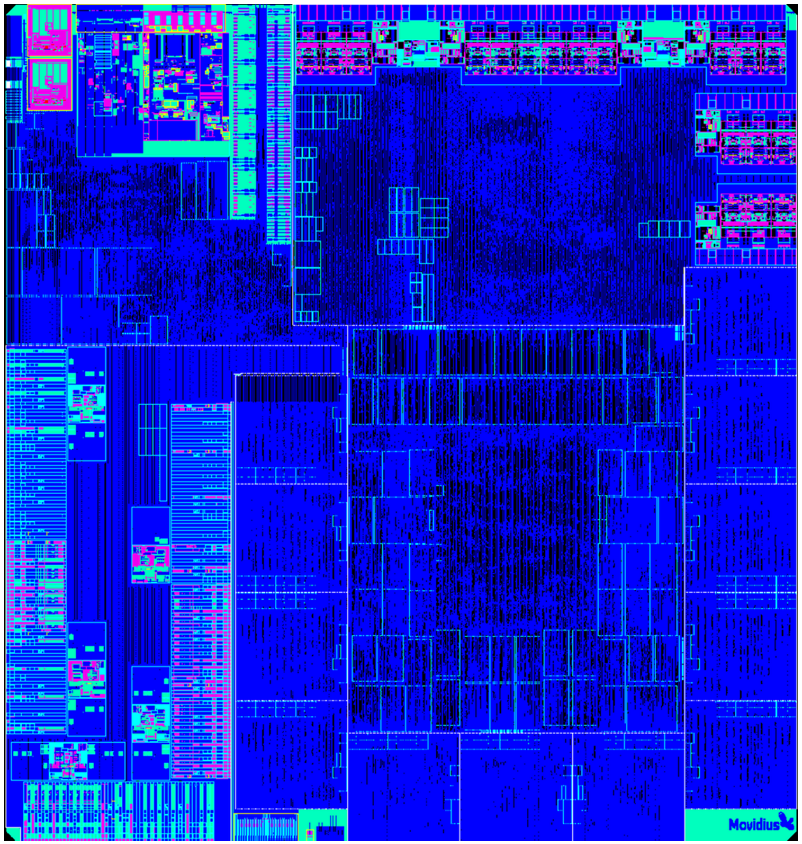


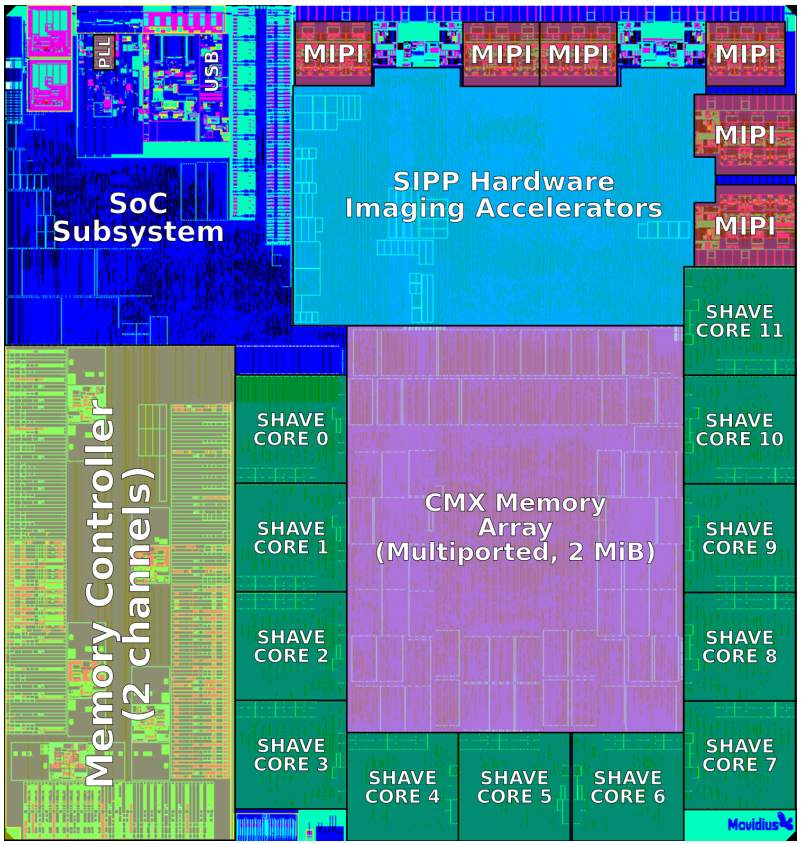
Introducing the Movidius Neural Compute Stick

- The NCS consumes about 1 watt.
- It achieves ~10x the performance of a CPU.
- Can be used to deploy already trained networks on edge devices.



Streaming Hybrid Architecture Vector Engine v3.0 (SHAVE v3.0)





Compatibility and usage

- Tensorflow
- Caffe
- mvNCCompile inception-v1.meta -s 12 -in=input -on=InceptionV1/Logits/Predictions/Reshape_1 -is 224 224 -o InceptionV1.graph
- mvNCCompile deploy.prototxt -w bvlc_googlenet.caffemodel -s 12 -in input -on prob -is 224 224 -o GoogLeNet.graph

The NCS only supports a limited subset of available layer types

- NxN Convolution with Stride S.
  - Group convolution
  - Depth Convolution
  - Dilated convolution
- Max Pooling, Average Pooling
- Local Response Normalization
- Relu, Relu-X, Prelu

The NCS includes firmware support for…

- Softmax
- Sigmoid
- Tanh
- Deconvolution
- Slice
- Scale
- ElmWise sum,prod,max
- Fully Connected Layers

The NCS includes firmware support for…

- Reshape
- Flatten
- Power
- Crop
- ELU
- Batch and L2 Normalization
- Input Layer

Some limitations

There are a number of limitations that could cause a network not to be compatible with the Intel Movidius NCS for reasons such as:

- Memory constraints.
- Layers not being supported.
- Unsupported neural network architecture.
  - RNN, LSTM

Luckily it's usually possible to get around these limitations.

Using the NCS part 1: Import and initialize graph (model)

```
# Convert an input tensor to 32FP data type
tensor = tensor.astype(numpy.float32)
# Write the tensor to the input_fifo and queue an inference
graph.queue_inference_with_fifo_elem(input_fifo, output_fifo,
tensor, 'user object')
# Get the results from the output queue
output, user_obj = output_fifo.read_elem()
```

Using the NCS part 2: Getting results from the network

The code below takes an input tensor (for example an image) and uses the trained model to obtain predicted output.

```
# Convert an input tensor to 32FP data type
tensor = tensor.astype(numpy.float32)
# Write the tensor to the input_fifo and queue an inference
graph.queue_inference_with_fifo_elem(input_fifo, output_fifo,
tensor, 'user object')
# Get the results from the output queue
output, user_obj = output_fifo.read_elem()
```

Using the NCS part 3:
Cleaning up device, fifo, and graph handles.

When your program reaches a state where the NCS is no longer needed (usually before exit), it is important to let go of the resources you allocated.

```
# Clean up
input_fifo.destroy()
output_fifo.destroy()
graph.destroy()
device.close()
device.destroy()
```

Using the NCS (full)

```
from mvnc import mvncapi as mvnc
device_list = mvnc.enumerate_devices() # Get a list of available device identifiers
device = mvnc.Device(device_list[0])# Initialize a Device
device.open()# Initialize the device and open communication
graph = mvnc.Graph('graph1')# Initialize a Graph object
input_fifo, output_fifo = graph.allocate_with_fifos(device, graph_file_buffer)
# Allocate the graph to the device and create input and output Fifos with keyword arguments
fifo_in, fifo_out = graph.allocate_with_fifos(device, graph_from_disk)
# Convert an input tensor to 32FP data type
tensor = tensor.astype(numpy.float32)
# Write the tensor to the input_fifo and queue an inference
graph.queue_inference_with_fifo_elem(input_fifo, output_fifo, tensor, 'user object')
# Get the results from the output queue
output, user_obj = output_fifo.read_elem()
# Clean up
input_fifo.destroy()
output_fifo.destroy()
graph.destroy()
device.close()
device.destroy()
```

Useful resources

https://www.tensorflow.org/guide/low_level_intro

https://movidius.github.io/ncsdk/ncapi/ncapi2/py_api/readme.html

https://aran.library.nuigalway.ie/handle/10379/6699

https://www.deeplearningbook.org/