Hands-on example: MNIST Inference with Arm NN

We show two fun digit recognition TensorFlow models doing inference with the Arm NN optimisation backend:

- a simple one layer neural network that barely recognizes anything
- a convolutional neural network that achieves close to 99% accuracy

Step 1. Connect to an Arm development board such as the HiKey 960:

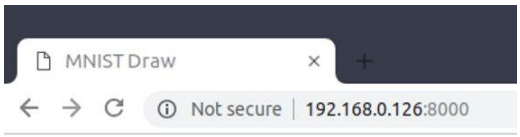
```
$ ssh arm01@<board-ip-address>
password is armml2018
$ ls
```

The development stack is assumed to be in the
$HOME directory, adjust paths as needed.

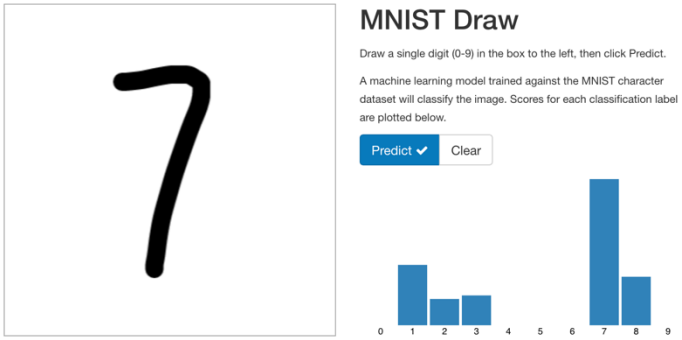
Step 2. Start the drawing canvas as a web server on the board

```
$ cd mnist-draw
$ python3 -m http.server --cgi 8000
```

Step 3. Point your local browser to the chosen port on the board →



Step 4. Draw a digit (neatly, please) and hope that the CNN model will recognize it



Step 5. Change the model to the Multi Layer Perceptron model and compare between the two

Change the parameter of the execution binary that the python script **cgi-bin/mnist.py** is calling:

```
# Run Arm NN model
try:
    # two integer arguments are:
    #   accelleration: 0 = CPU unoptimized, 1 = CPU accelerated, 2 = GPU accelerated
    #   model: 0 = simple, low accuracy  1 = optimized, higher accuracy
    #     change the 2nd 1 to a 0 to use the low accuracy model
    completed = subprocess.run(['./armnn-draw/mnist_tf_convol', '1', '1', 'image.txt'],
                stderr=subprocess.PIPE, check=True)
except subprocess.CalledProcessError as err:
    print('ERROR:', err, file=sys.stderr)
```

This fun application demonstrates that even MNIST inference is not trivial. There is trade off between performance and accuracy, but a fast model with mostly wrong answers is not fun.