TTN Conference UK workshop: ML on MCU, building a Gesture Pod.

Workshop summary

This workshop is built using the EdgeML project from Microsoft Research. The full EdgeML repository is available at https://github.com/microsoft/EdgeML

It's purpose is to demonstrate the ability to create and run a small deep learning solution on a constrained microcontroller device.

The workshop covers the assembly of a constrained device to which a deep learning model is deployed to evaluate the output of a motion processing unit to recognise a small set of gestures. The trained model using EdgeML's ProtoNN for 6 gesture classes is only 6.5KiB in size.

This workshop guide also covers process for creating a new gesture model using the Pytorch implementation of a ProtoNN trainer. This allows workshop participants to customise their gesture model to suit needs, providing the basis for a smart connected device capable of determining real world movement and reporting it to a larger system.

Workshop participants should complete the pre-workshop machine setup process on the next page before attending the workshop

The constrained device hardware used in this workshop is:

Adafruit Trinket M0

- 32 bit Cortex M0+
- 256kb Flash
- 32Kb RAM
- 48 Mhz

InvenSense MPU-6050 breakout board

- 3 axis gyroscope and 3 axis accelerometer with a Digital Motion Processor
- Supports I2C communications: up to 400 kHz
- User-programmable gyroscope full-scale range of ±250, ±500, ±1000, and ±2000°/sec (dps)
- User-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g

Pre-workshop machine setup

Prior experience shows that attempting to set up your machine for the workshop during the workshop or at the conference venue will be challenging and wasteful of your time and the workshop organisers. Please setup your machine prior to attending the workshop so you can have the best workshop experience.

Installation requirements:

1. Setup the Arduino IDE editor for your OS:
   Arduino IDE available from here (use installed versions not web client) https://arduino.cc
   Set up device and library dependencies:
   a. The Adafruit Trinket M0 device: https://learn.adafruit.com/adafruit-trinket-m0-circuitpython-arduino/arduino-ide-setup
   b. The I2CDevLib: https://www.i2cdevlib.com/usage

2. Install Anaconda or other python 3 development environment. Required for creating your own gesture model.

   Anaconda download and installation: https://www.anaconda.com/distribution/
   Select your OS installer, download and run.
   Setup a virtual environment for EdgeML installation: (Anaconda).

3. Clone repo

   Using your GitHub tool of choice clone the https://github.com/PaulDFoster/GesturePodWorkshop repo or download it as a ZIP file and extract it in a suitable place on your machine.

4. Setup the Pytorch model generation environment:

   Change directory to <location of GitHub clone>\EdgeML\pytorch

   Use pip and the provided requirements file to first install required dependencies before installing the pytorch_edgeml library.
   It is highly recommended that EdgeML be installed in a virtual environment. Please create a new virtual environment using your environment manager (virtualenv or Anaconda). Make sure the new environment is active before running the below mentioned commands.

   CPU

   ```
   pip install -r requirements-cpu.txt
   pip install -e .
   ```

   Tested on Python 3.6 with PyTorch 1.1.

   GPU

   Install appropriate CUDA and cuDNN [Tested with >= CUDA 9.0 and cuDNN >= 7.0]

   ```
   pip install -r requirements-gpu.txt
   pip install -e .
   ```

   Note: If the above commands don't go through for PyTorch installation on CPU and GPU, please follow this link.

Assemble the basic Gesture Pod device

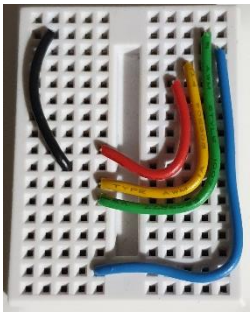
Bill of materials:

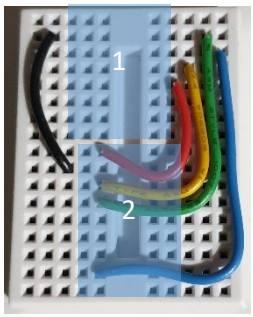
1. Adafruit Trinket M0
2. MPU-6050
3. Mini breadboard
4. Pre-cut jumper wires
5. USB cable (please use your own, limited number available in workshop)

The hardware in this workshop has be funded by the workshop leader, please return before leaving the workshop. The hardware is available to purchase at minimum cost price of £10 if you wish to keep it.

Assembly:

Assemble your device using the following diagram:





1. Trinket M0, 2. MPU-6050

Wire	Trinket M0	MPU-6050
Black	GND	GND
Red	3V	VCC
Yellow	2	SCL
Green	0	SDA
Blue	1	INT

Deploy example solution and test

Plug the Adafruit Trinket M0 into your PC USB.

In the Arduino IDE setup the Adafruit Trinket M0 as the board and select the correct port (on Windows this will be named correctly)

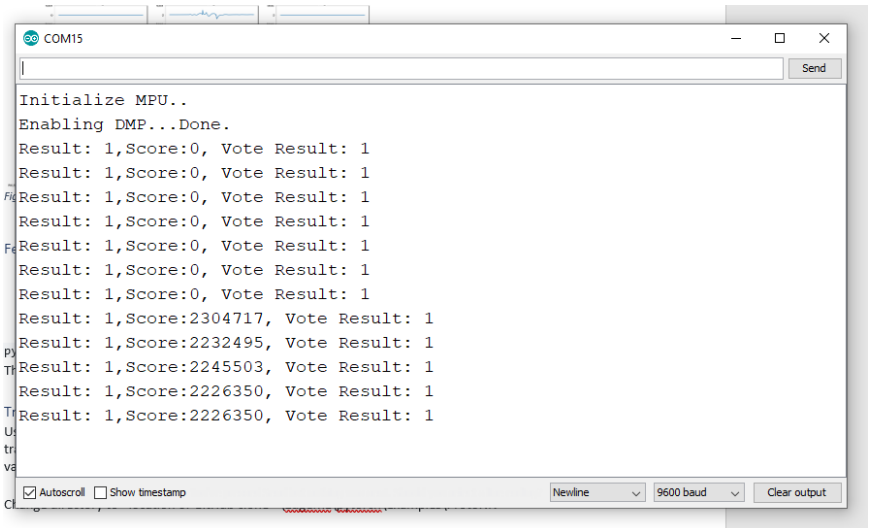
Open the GesturePod.INO file from cloned GitHub repo.

Build and deploy the sketch to your device.

Troubleshooting:

If the build fails, check you have the dependant libraries installed, the correct board and COM port selected.

Testing: Open the Serial Monitor (set to 9600 baud). Correct output will be:



RGB Led.

The RGB Led on the Trinket will be used to indicate the recognised gestures. The LED starts off violet when the Trinket is reset.

Gestures and LED colours:

Gesture	LED Colour
Double Tap	Red
Right Twist	Green
Left Twist	Blue
Twirl	Orange
	Pink
Double swipe	Yellow

Training ML model for gesture recognition

Introduction

It is not expected that workshop participants will be able to complete the training of a new model within the time limit of the workshop. It is hoped that this guide will enable the process to be completed post-workshop.

Training a model to recognize gesture on the GesturePod consists of 5 parts:

1. Data collection
2. Data labelling
3. Extracting the features from the labelled data
4. Training a ML model
5. Deploying the model on the GesturePod to recognize gestures in real life.

Data Collection

Accelorometer and gyroscope values from the MPU6050 sensor is collected. Connect the GesturePod to computer over Serial COM Port.

1. Compile, Build and Upload Navigate to <location of GitHub clone>\EdgeML\Applications\GesturePod\training\serialDataCollection upload serialDataCollection.ino onto the GesturePod. Make sure you select the COM port to which Trinket M0 is connected.
2. Launch the COM port and verify that data is being received. It should be of the format [time, acc_x, acc_y, acc_z, gyr_x, gyr_y, gyr_z].

3. On a python3 environment run the following command to save the raw data:

```
python csvFromSerial.py [outputFileName] [COMPort]
```

The raw data file is stored at ./data/raw_data/.

With the data capture running, perform 5 to 10 gestures with a 2 second gap between them (I found counting – “1 elephant, 2 elephant” provided good spacing.) per data file. Close the capturing python script by typing a ‘q’ and pressing return. You should aim for 60+ examples of the gesture. The more complex the gesture, the more data you need to define it in the model.

Capture some noise data – that data which is not your gesture movements. Place this in a noise csv file for using in the model generation stage. Noise files do not need to be labelled.

Data Labelling

1. The raw data in `./data/raw_data/foo.csv` must be labelled with the following command:

```
python labelData.py foo.csv
```

This command will launch an interactive GUI. Sensor values will keep rolling across the screen. Use spacebar to pause the action. Center the signature using the arrow keys, and key in the corresponding "label" using number keys. Press spacebar to continue. For example, on seeing a signature of Double Tap - hit spacebar to freeze the values. Then after centering the signature using right/left arrow keys, hit 3 (DTAP). Press spacebar to continue.

The labelled data file `foo_labelled.csv` will be written at `./data/labelled_data/`.

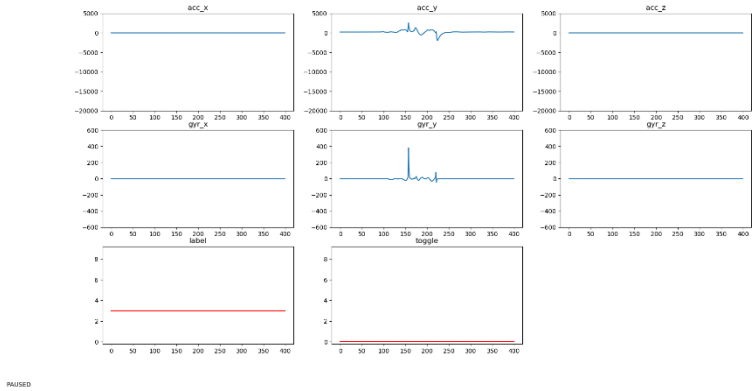


Figure 1 Example labelling screen

Feature Extraction

1. Enlist the labelled files in *labelledFileList* List in generateFeatures.py. Files that do not contain any gestures - for example, data of climbing stairs, walking in park, etc should be listed in *allNoiseFileList*.

2. Generate the set of features for labelled data.

```
python generateFeatures.py
```

This will generate a `train.csv` and `test.csv` files that should be used to generate a ML model.

Training and Deploying

Using the PyTorch implementation of the ProtoNN algorithm from the EdgeML repository, train a model on the train.csv file generated above. Extract W, B, Z, and gamma values from the trained ProtoNN model. Update these values in `EdgeML/Applications/GesturePod/onMKR1000/src/data.h` to deploy the model on the GesturePod.

Change directory to <location of GitHub clone> \EdgeML\pytorch\examples\ProtoNN

A Jupyter Notebook protoNN_example.ipynb or a command line script protoNN_example.py are available to train a model using the pytorch protoNN algorithm implementation. Jupyter Notebooks can be started from the Anaconda command line by: jupyter notebook

Command line script execution:

```
python protoNN_example.py \
      --data-dir ./mpu \
      --projection-dim 10 \
      --num-prototypes 20 \
      --gamma 0.0014 \
```

```
    --learning-rate 0.01 \
    --epochs 200 \
    --val-step 10 \
    --output-dir ./
```

Either approach will result in W,B and Z files being created in the directory.

The values of these files can be copied and pasted to replace the corresponding array values in the Arduino GesturePod sketch data.h file found in the src subdirectory.

A data.h file can be automatically generated from the W,B and Z files using the following command:

python gen_data.py [gamma_value]

For example: *python gen_data.py 0.0014*

Copy the created data.h file to the src directory of the GesturePod Arduino sketch – you may wish to backup the existing data.h file so you can return to using it later.

Compile and deploy the new GesturePod sketch to the Trinket M0. See if your model recognises your gestures.

Trouble shooting:

Why does it not recognise some/all of my gestures:

- The model was made without enough example gestures captured. Try capturing more data and regenerating the model using it and your existing training data.
- The gestures do not have a clear signal in the data. Try different or more expansive gestures
- The gestures do not cause data signal on the Y axis. The current GesturePod implementation analyses only Y axis data.

Dependencies

To communicate with the MPU6050, we use jrowberg's i2cdevlib library. Last tested with commit 900b8f9.

EdgeML and Gesture Pod

EdgeML is a Microsoft Research project which has created several new algorithm implementations for constrained devices, including ProtoNN. The source code and research publications are all available from the team's GitHub repository: https://github.com/microsoft/EdgeML

The model training materials are published off the model_training branch at this time, and are not in the master branch.

A video of GesturePod in use is available here: https://1drv.ms/u/s!AjDloPaG_l0Et7Ikid1voOVFuI116Q