Lab 2 - Motion Classification

Introduction

We will ask you to create a fully functioning embedded machine learning system in this lab. The process will include collecting data, performing feature extraction, training a model, and deploying that model to an embedded system.

We will classify the motion and vibration data made by you (as in the class example) or from a machine of your choice. This is to mimic using embedded machine learning in an industrial environment. We want to determine if a device is off, low, high, anomaly, etc.

Required Hardware

You should use the TinyML Kit based on Arduino Nano 33 BLE Sense to complete the project. You will also need some tape (recommended: electrical tape) to secure your Kit to the machine (If you prefer, use only the Nano-33 instead the complete Kit.

Setup

Before collecting data, we must first figure out what we want to monitor! We constructed a demo that simulates cargo (container motion) in the class. In this project, we will work preferably with machines to classify operating modes and look for anomalies (As a minimum task, you can repeat the class example with different movements/gestures/motion).

First, choose a machine in your home that produces some vibration data that you wish to monitor and classify. Here are some ideas of things you may want to watch:

- Washing machine
- Blender
- Air compressor
- Refrigerator
- Air conditioner
- Keyboard (typing)
- Ceiling fan

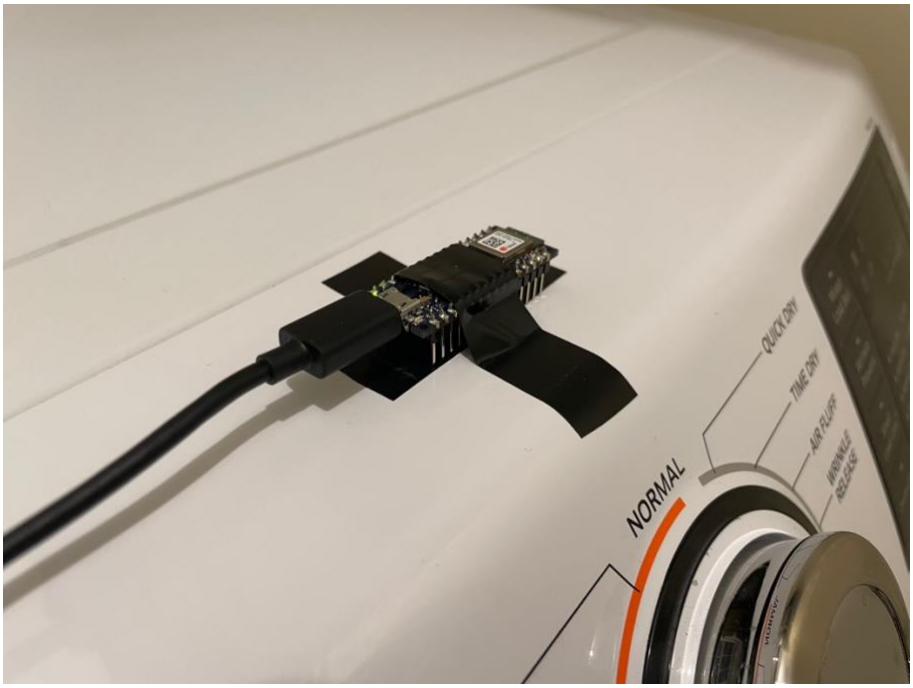
Connect Device to Edge Impulse

Create a new project on Edge Impulse. Give the project a name, such as “IESTI01-industrial-motion-classifier.”

Attach Device

Use some tape to adhere your Arduino board (or Kit) to your chosen device.

WARNING: If the device is metal, put a layer of tape down as a barrier to prevent the Arduino’s pins from shorting together. Also, take care of handling water or liquids near the Arduino.



Note that you will need to connect your Arduino board to your computer. Connect your Arduino to your computer using a USB cable.


SAMSUNG
SAMSUNG


Oster

Data Collection

Head to the **Data acquisition** page in your Edge Impulse project. Your device (if connected) should show up on the right side of the window. If not, follow the steps shown in class to connect a device to your project.

You will need to choose a few classes for your model to be able to predict. For example, on the above “Blender project,” I tried to capture the motion on a 12 Speed Blender (with a pulse button).

The classes were:

1. Off (motor off, with eventual movent of the blender)

2. Low_Empty (Speed 1, Empty Jar)

3. Low_Water (Speed 1, Jar with 500ml of water)

4. High_Water (Speed 12, Jar with 500ml of water)

For the Washing Machine project, the model was able to identify when the dryer was in one of the following states:

- Off
- Light load
- Heavy load

Collect at least 2 minutes of accelerometer data for each class. You should see the collected data filled out with samples when you're done. Don’t forget to label your class before you start collecting data!

When you’re done, repeat the process for the test data. However, you only need to collect about 30 seconds of data for each class.

Feature Extraction

Head to the **Impulse design** page in your project. Add a **Spectral Analysis** processing block. Add a **Neural Network** block and a **K-means Anomaly Detection** block to the learning blocks section.

Head to the **Spectral Features** page and click **Generate features**. Take a look at the *Feature Explorer*. What patterns do you see in the samples? Can your classes be easily separated?

Try changing the X-Axis, Y-Axis, and Z-Axis parameters in the *Feature Explorer*. Which features offer the best separation among the groupings of labels? In our class example, RMS showed decent separation. Write down which set of features provides the best separation. You will want to select these features in the Anomaly Detection section.

Model Training

Head to the **NN Classifier** page in your project. Click **Start training**. After a few minutes, the model should be done training.

Scroll down and take a look at your confusion matrix. You should see good accuracy for the whole model and within each class.

If you're not happy with the results, try changing some hyperparameters and train again! For example, change the training cycles (epochs), add another neural network layer, change the nodes in each layer, learning rate, etc.

Be careful of overfitting! If you see that the validation loss is much higher than the training loss (look carefully at the training output), your model may be overfitting. In that case, try reducing the number of nodes or layers in your model and try again.

The ultimate goal is to find the smallest neural network (smallest number of nodes and layers) that meets your needs. As you increase the number of nodes and layers, you increase the computational complexity of the model, which requires more resources in your microcontroller.

Ideally, you want your validation accuracy to be 80-95%, depending on your application.

Go to the **Anomaly detection** page of your project. Select the features you wrote down in the previous section (these will most likely be the RMS values for the X, Y, and Z axes). Click **Start training**. You should see your features with their clusters in the Anomaly Explorer when that's done.

Model Testing

Go to the **Model testing** page in your project. Click the checkbox next to *Sample Name* to select all of your test samples. Click **Classify**. After a few moments, your test set should be classified using the model you trained.

If you see something less than 65% accuracy, your model may be overfitting or underfitting. Try collecting more data and adjusting some of the hyperparameters.

Be careful! Each time you update your model's hyperparameters because of some new information from the test set, you begin introducing bias into your model. It may overfit the test set as well. At this point, your test set is no longer a test set but rather a secondary validation set.

I recommend gathering new test data if you plan to adjust the hyperparameters and re-train the model at this point. You can move your test samples to the training set (in *Data acquisition* > *Test data*). Also, go to the Dashboard for your project. You can click *Rebalance dataset* to have Edge Impulse automatically group your training and test datasets together and then randomly split them up again into training and test sets (Warning: Check if the train and test datasets are balanced. If not, proceed with a manual split).

Deployment

Ensure that your Arduino board is in the same spot as when you collected data! If you move it, you could affect the accelerometer readings.

Head to the **Deployment** page in your Edge Impulse project. Click on **Arduino library**, scroll down, and click **Build**.

Once your project has been downloaded, open the Arduino IDE. Go to **Sketch > Include Library > Add .ZIP library**. Select your downloaded .ZIP file from Edge Impulse. Go to **File > Examples > <your-project> > nano_ble33_sense_accelerometer**.

Click **Upload** to compile and send the program to your Arduino board. This may take over 10 minutes to compile and upload.

When it's done, open a serial monitor. Run your machine in the various modes to see if your Arduino can correctly classify the vibration mode.

Post-Processing:

Try flashing the Arduino's onboard LED whenever a particular class or anomaly value is above a threshold. What threshold you choose is up to you. I recommend starting with 0.6 if you're using a class label or 0.3 using the anomaly score.

For example, in the blender project, the Arduino code was changed to include the Nano LEDs, following the logic:

- High_Water ==> Red ON
- Low_Empty: ==> Green ON
- Low_Water: ==> Blue ON
- Off: ==> All OFF

If the anomaly score was positive, the internal LED was also ON

- Anomaly ==> LED_BUILTIN ON