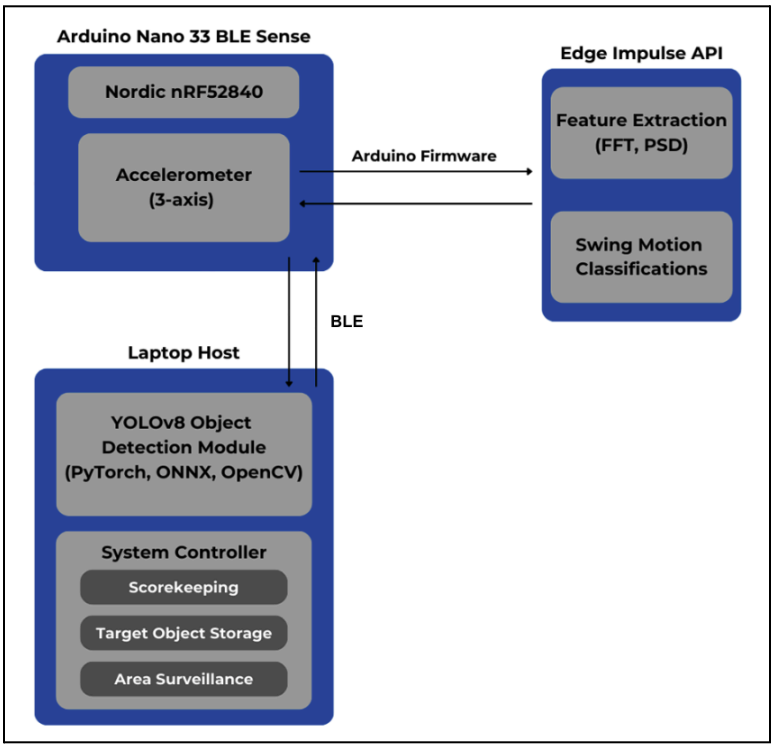
18-448 TinyML Baseball Simulator Final Report

Belle Connaught, Elliot Clark

1. Abstract

Our project implements a real-time baseball detection and scoring system using YOLOv8 for object detection and Bluetooth Low Energy (BLE) for external device communication. The system integrates a webcam to capture live video frames, processes them with a pre-trained YOLOv8m model to detect baseballs, and overlays detection results with bounding boxes and confidence scores. Simultaneously, a BLE client receives predictions from an Arduino-powered external device, enabling enhanced interaction and data integration. A scoring mechanism updates dynamically based on baseball detections and BLE input, providing a seamless user experience. Our solution demonstrates the potential of combining modern computer vision and BLE communication for interactive sports analytics.

2. Block Diagram



3. Dataset Sourcing

For dataset creation, we collected a total of 200 samples each for diagonal, vertical, and horizontal swings, as well as an additional 50 samples for the idle class. To acquire each sample, we first connected the Arduino Nano 33 BLE Sense to Edge Impulse. We then set the window size to 1.5s, and executed the appropriate motion (at a constant, relatively moderate speed) while holding the Arduino in the palm of our hands. During the cleaning process, we identified and removed examples that appeared to be outliers, such as instances where the motion did not conform to the expected patterns or contained significant noise. This step

ensured the dataset's quality and consistency, providing a reliable foundation for training and evaluating the system's motion classification performance. We also cropped each sample to be exactly one second long, removing potential noise at the beginning and end of the recording.

4. Feature Extraction

We focused on spectral features to capture the frequency-domain characteristics of the motion data, as we did in the Motion Classification assignment. Using Fast Fourier Transform (FFT) with overlapping frames, we analyzed the accelerometer signals to extract meaningful patterns that differentiate between diagonal, vertical, and horizontal swings, as well as the idle class. We also increased the FFT frame length from 32 to 128 samples, as we thought this would lead to improved frequency resolution and capture more detailed patterns in the motion data. This adjustment proved particularly effective for improving the model's ability to distinguish between *vertical swings* and *diagonal swings*, where subtle differences in frequency content had previously led to slight classification difficulties.

5. Classifier Architecture

We experimented with different classifiers and ultimately selected the one that provided the best balance of accuracy and performance for our dataset. We ultimately settled on a neural network trained over 50 training cycles, with a learning rate of 0.005 and two 20-neuron dense layers between the input and output layers. The model initially had only 10 neurons in the second layer, but the resulting accuracy (~89%) and confusion matrix revealed that it was overfitting slightly. We then tried decreasing the learning rate to 0.004 or 0.001, increasing the number of training cycles to 60 or 70, and a combination of each method. However, our training accuracy would not go above 90%, and our test accuracy hovered around 88%. We then found that increasing the second dense layer to 20 neurons increased our training and test accuracies to 91.1% and 93.98%, respectively. This is because increasing the layer size increases the non-linearity of the model, allowing it to generalize without overfitting so much since each additional neuron has the capacity to capture a different feature or pattern.

6. Confusion Matrix and Accuracy

Training Performance:

ACCURACY
91.1%

LOSS
0.09

Confusion matrix (validation set)

	DIAGONAL SLASH	HORIZONTAL SLASH	IDLE	VERTICAL SLASH
DIAGONAL SLASH	90%	10%	0%	0%
HORIZONTAL SLASH	6.7%	86.7%	0%	6.7%
IDLE	0%	0%	100%	0%
VERTICAL SLASH	0%	11.1%	0%	88.9%
F1 SCORE	0.92	0.84	1.00	0.89

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	0.91
Weighted average Recall	0.91
Weighted average F1 score	0.91

Testing Performance:

ACCURACY
93.98%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	0.95
Weighted average Recall	0.95
Weighted average F1 score	0.95

Confusion matrix

	DIAGONAL SLASH	HORIZONTAL SLASH	IDLE	VERTICAL SLASH	UNCERTAIN
DIAGONAL SLASH	85.7%	4.8%	0%	0%	9.5%
HORIZONTAL SLASH	4.5%	95.5%	0%	0%	0%
IDLE	0%	0%	100%	0%	0%
VERTICAL SLASH	0%	3.6%	0%	96.4%	0%
F1 SCORE	0.90	0.93	1.00	0.98	

7. Deployment Method

To deploy our model, we used Edge Impulse's Arduino Library with an Arduino Nano 33 BLE Sense as the target device. We then configured the resulting .ino sketch file, initially in a similar manner to the Motion Classification assignment: initially, the sketch only processed accelerometer data, but we expanded it to also include gyroscope data by adjusting the buffer size and indexing. Specifically, we ensured that the buffer could store both accelerometer and gyroscope readings by re-indexing the data points and adding the necessary lines of code to read the gyroscope sensor at the end of the buffer. To ensure data quality, I added a range check that capped any readings exceeding the predefined maximum range (±2g for accelerometer data), and then converted the accelerometer values to meters per second squared (m/s^2) using the CONVERT_G_TO_MS2 constant (whose value was 9.80665f).

Additionally, we made modifications to the smoothing function. Initially, the predictions were not consistently stable, so we adjusted the smoothing settings to improve prediction accuracy and reliability. I

configured the smoothing function to require predictions to match for 10 out of 20 frames before they would be considered valid, with a minimum confidence level of 0.7. This adjustment ensured that predictions were only transmitted when sufficiently confident, helping to reduce false positives or misclassifications.

We also used the AduinoBLE library to integrate BLE communication, allowing us to transmit each prediction over a 32-byte characteristic to a laptop, which contained our computer vision code. Essentially, using multitasking via the **asyncio** library, we implemented a system in which a YOLOv8m model continuously analyzes webcam video frames to detect baseballs, which fall under the "sports ball' category. When a baseball is detected, the system checks if a non-idle prediction was received from the Arduino within the past 5 seconds, then increments the score and prints the prediction to the terminal.

8. Challenges & Lessons Learned

Our project taught us to pay close attention to the limiting factors of our design; integrating the CV2 object tracking together with the incoming BLE data was not problematic, but the series of sensor inputs introduced a noticeable delay to the final product. It was because of this latency that we reworked our project idea to involve the target object starting in the camera frame rather than entering view mid-air. The timing-intensive operation of the system also required a fast response from the Arduino board–about 1s maximum–to accurately register against baseballs in the air, which was not fully achievable. Regardless, keeping these concerns in mind helped us to think more deeply about the design process, and a solution for lowering the response time could certainly be achieved given more time.