NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Automatic Mini Basketball Scoring System

CE4172 Internet of Things: Tiny Machine Learning

Wong Wei Bin (U2022892D)

19 Apr 2024

1. Introduction & System Overview

Background and Objective

The free throw is a fundamental shot in basketball that players train for years to perfect. It is the most consistent shot, taken from a stationary position the same distance away from the basket every time. However, when practising alone, players may find it difficult to keep track of their performance and accuracy after taking several shots. It is hence a challenge for players to review their performance to find areas of improvement or get feedback in real time when practising the shot. The only alternative is for the player to video record the whole training session and watch it afterwards, which is very inefficient.

The consistency of the shot and the need for a real-time feedback and post-training review system opens the opportunity for the use of IoT devices and embedded machine learning to create an automatic scoring system.

System Overview

This project uses an Arduino Nano 33 BLE Sense, which has sufficient computational power to handle real-time data processing, and is compact enough to be attached to a basketball net. The Arduino Nano also has a built in Inertial Measurement Unit (IMU), which was used to capture accelerometer and gyroscope data to train a model to predict whether a shot was a make or miss based on the net and rim's movement and record it. For software, Python was used for data handling and model training. Arduino IDE was used to program the Arduino Nano for both data collection and real-time inference. Finally, MIT App Inventor was used to develop a mobile user interface for the user to view the shot records in real time.

This project is a proof of concept, and uses a toy mini-hoop and ball in place of a real-life basketball hoop, for greater ease and accessibility of data collection and testing. The evaluation and viability of this proof of concept will be covered in the conclusion section of this report. Figure 1 below shows the final product of this project.

Figure 1: Final Product - Mini Basketball Hoop Integrated with Arduino Nano 33

2. Methodology

Data Collection & Processing

The data collected is sequential time series data. Each shot attempt is characterised by the IMU data of the Nano attached to the net, resulting from any movement or vibration from the ball's impact, starting from the point of impact and lasting a set period of time until the hoop settles back to a stationary position.

To collect data, an arduino program imu_ball_capture.ino was written and uploaded onto the Arduino. It sets a trigger threshold that, when exceeded, captures IMU data for 1.5s and displays it on the serial monitor. The trigger threshold is low enough such that it is exceeded the moment the ball hits any part of the rim, backboard or net. To speed up data collection, a python script shot_data_collection.py was used to read the data on the serial monitor and save it to either a made_shots.csv or missed_shots.csv file based on the outcome after each shot. Data for 200 shot attempts, 100 makes and 100 misses, were collected.

Figure 2 below is a screenshot of the csv data collected. The data from both files was combined into a dataframe, labelled with one-hot encoding, normalised, randomly shuffled and split into train and test sets (80-20) prior to model training.

	A	B	C	D	E	F	G	H
1	ShotID	Time	aX	aY	aZ	gX	gY	gZ
2	2	0	1.676	-0.495	-1.477	-184.937	85.51	-65.308
3	2	1	0.246	0.147	-0.116	-506.531	150.94	-133.972
4	2	2	-0.036	0.239	0.024	-346.68	93.262	-113.403
5	2	3	-0.6	-0.037	-0.65	234.802	-2.502	-109.741
6	2	4	-0.709	-3.498	-4	1379.456	-177.856	-90.088
7	2	5	-0.515	-4	-4	1869.019	-667.969	146.301
8	2	6	-0.091	-1.042	-1.936	1295.044	-1289.001	369.995
9	2	7	-4	-4	4	1048.889	-1020.935	248.779
10	2	8	3.262	-2.715	-4	-305.969	-61.89	162.17

Figure 2: Screenshot of made_shots.csv

Model Training

The two models that were looked into for this project were a Long Short-Term Memory (LSTM) model and a 1-Dimensional Convolutional Neural Network (Conv1D) model. LSTM was chosen due to its proficiency in handling sequential and time-dependent data. LSTMs are ideal for learning from time-series data as they capture long-term dependencies for accurate real-time predictions. Moreover, as of 8th September 2022, the tflite-micro library was updated to support the use of unidirectional sequence LSTMs, which was long awaited by the community.

Conv1D models were also trained to benchmark the performance of the LSTM model. Conv1D is another common model for analysing time-series data that uses sliding windows to extract features from sequences, offering a fast and parameter-efficient alternative to LSTMs. LSTMs are best suited for applications that require analysis of long-term dependencies in time series data whereas Conv1D models are more efficient for taks that focus on identifying short-term local patterns within sequences. From a theoretical standpoint, Conv1D will be the more suitable choice for the short-term nature of the data used in this project, ranging across a time period of only 1.5s per batch.

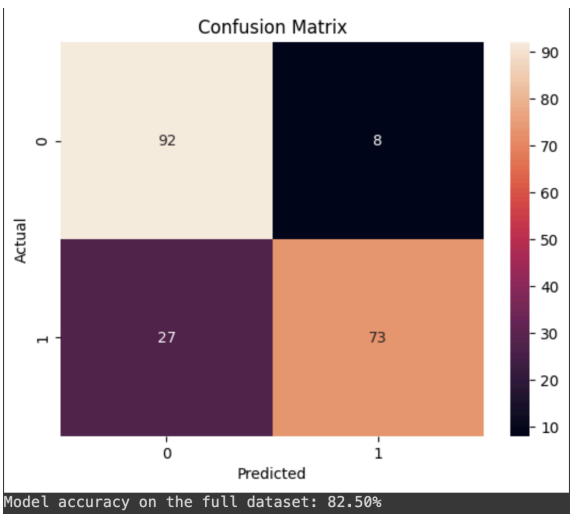
A simple single-layered 8 neuron LSTM was first used to gauge model accuracy. It was observed that the model performed better with more neurons in a single layer, and even better with dropout and a second LSTM layer. Dropout is a regularisation technique that prevents overfitting in neural networks by randomly setting a fraction of input units to zero during training, which helps improve model generalisation and accuracy on unseen data. The highest accuracy observed when tested on the whole dataset was 82.5%.

To provide comparison, a single layer Conv1D model with 64 filters and max pooling was also trained. It had much faster training times than the LSTM models and achieved higher accuracies. Adding one more Conv1D layer with 32 filters made the model even better, reaching an accuracy of 96.5% when tested on the whole dataset. This model had the highest train set, test set, and overall accuracy of all models trained.

TFLite Conversion & Model Selection

After training, each model was converted into a TensorFlow Lite (TFL) format using the TFLiteConverter. It applies quantisation optimisation to reduce the model's size and computational demand, making it suitable and efficient for deployment on the resource-constrained Arduino Nano. For conv1D models, a representative dataset generator was used to specify a representative dataset for calibration, in order to perform full integer quantisation. If this is not done, the TFLite-micro library will recognise the TFLite conv1D model as a hybrid model with different operations using different numeric data types and throw an error.

However, for LSTM models, a large drop in accuracy (up to 30%) after TFL conversion was repeatedly observed. This was not seen in the conv1D model, which posted almost the exact same accuracy and confusion matrices before and after conversion. Figure 3 and 4 shows this comparison.



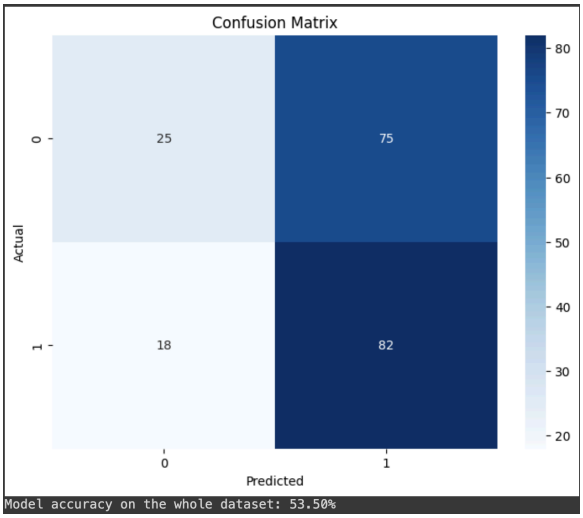
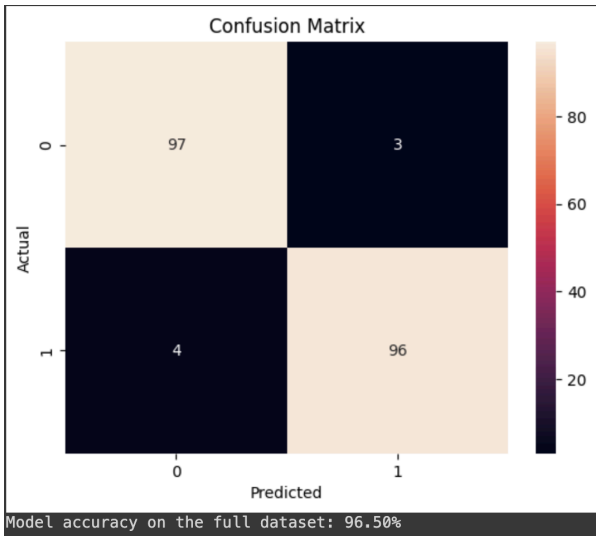


Figure 3: Confusion Matrices for LSTM Model (1 layer) Before (left) and After (right) TFLite Conversion



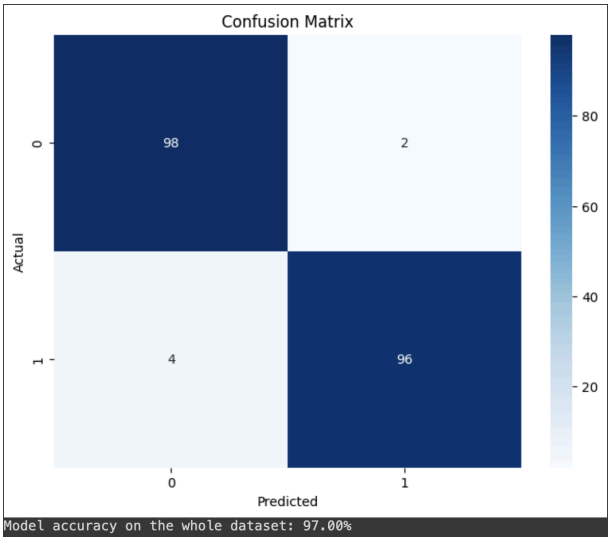


Figure 4: Confusion Matrices for Conv1D Model (2 layers) Before (left) and After (right) TFLite Conversion

This is likely because conv1D models are generally simpler than LSTM models, which have more complex operations that are only partially supported or optimised in Tensorflow Lite. Besides, the precision loss during quantisation can significantly impact LSTM performance due to its sensitivity to small changes in weight values. The table below summarises the model accuracies and size after encoding to C for all the models trained.

Model	Parameters	Best Accuracy on Whole Dataset		Model Size (Encoded to C)
		Before Conversion	After Conversion	
Conv1D	2 layers, 32 & 64 filters, with max pooling	96.5%	97.0%	119KB
Conv1D	1 layer, 64 filters, with max pooling	94.5%	94.5%	106KB
LSTM	2 layers, 16 & 8 neurons, with dropout	78.0%	78.5%	85KB
LSTM	1 layer, 16 neurons	82.5%	53.5%	56KB

The 2 layered Conv1D model emerges as the most compelling choice for deployment. Despite it being the largest in size, it still fits well within the capacity of the Arduino Nano, which has 1MB of flash memory for storing the program, including the TFL model. Furthermore, the operations utilised by the model, such as Conv2D and others depicted in the architecture visualised through Netron (Figure 5), are fully supported by the TensorFlow Lite Micro Library, as included in the tensorflow/lite/micro/micro_mutable_op_resolver.h file. This Conv1D model with 97.0% accuracy after conversion was the model that was uploaded onto the Arduino for deployment.

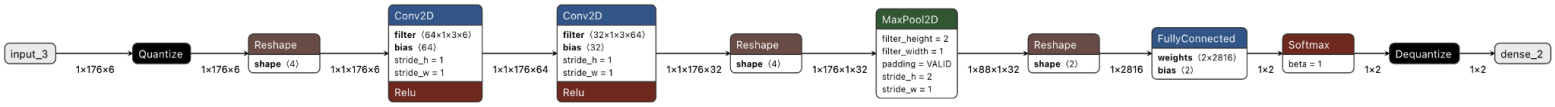


Figure 5: Conv1D (2 layers) Model with Layers Represented as Operations

3. Implementation and Optimisation

Code Implementation

After the Conv1D model was converted into TensorFlow Lite format and encoded as a header file, it was included in the Arduino programming environment. A sketch in the same environment was developed to be uploaded onto the Arduino Nano during deployment. The sketch includes code to establish a Bluetooth connection, load the model, set up the TFLite interpreter and allocate memory. Upon detecting motion exceeding a trigger threshold, IMU data is captured over a set duration, like in the data collection program. This data was normalised and fed into the TensorFlow Lite model, by writing to its input tensors. It is then invoked to perform inference and the prediction results are written to the output tensors. The predictions, quantified as confidence levels for each class (made or miss), were then labelled as 0 or 1, based on a confidence threshold of 0.5. This predicted label is transmitted via Bluetooth to an Android app interface created using MIT App Inventor.

App Functionality

The Android app acts as a client to connect to the Arduino. The user would scan for discoverable bluetooth devices on the app, select the Arduino Nano, and click on the ‘Connect’ button to establish a connection, or the ‘Disconnect’ button to disconnect anytime.

The app displays the inference results in real-time with each shot attempt. It also logs cumulative shot data - shots made, missed, total shots, and shooting accuracy percentage over a period of 60 seconds, represented by a countdown timer. This real-time recording begins when the user clicks on the 'start' button and can be reset at any point using the 'reset' button. The inference stops when the timer runs out and is only restarted when the user clicks on the ‘reset’, followed by the ‘start’ button. The app interface is shown in Figure 6.

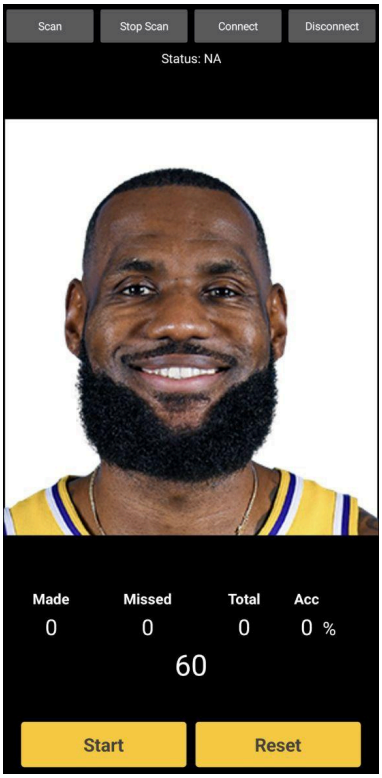


Figure 6: Screenshot of App Interface

Optimisation for Performance

The system is designed for power efficiency. The Arduino Nano remains in a low-power state upon startup, awaiting a Bluetooth connection from a client before initiating any data capture or model inference. This ensures that power consumption is minimised when the device is not actively processing shots.

Moreover, the Arduino only performs inference when the IMU's trigger threshold being surpassed, preventing unnecessary computations during periods of inactivity, such as when the user is preparing for the next shot.The integration of Bluetooth Low Energy (BLE) allows for efficient transmission of data between the Arduino and the app, with the Arduino programmed to send inference results to the client following each shot. The client app, which connects to the Arduino via BLE, has been optimised to handle these incoming data packets effectively, updating the shot results and accuracy metrics in real-time.

By leveraging the TensorFlow Lite Micro library, the model's operations are fully compatible with the microcontroller's capabilities, enabling seamless real-time inference. This compatibility, along with the aforementioned power efficiency strategies and the Bluetooth integration for data communication, culminates in a robust system for real-time basketball shot tracking and analysis.

4. Results and Conclusion

Inference Performance Results

The model is expected to have a 97% accuracy, based on the confusion matrix (Figure 4) and accuracy on the whole dataset calculated after conversion to TFLite format. However, upon testing the whole system after deployment, the actual prediction accuracy is estimated to be around 80-90% (1-2 prediction errors per 10-14 shots). This is likely because the model was only trained on 200 shots, missing out data on shots that occur less frequently. For example, shots where the ball bounces on or rolls around the rim before dropping through the net, are often classified wrongly by the model, as the data points for these shots were either under represented or not represented at all in the dataset.

Moreover, shots with different outcomes may record similar movement profiles and hence similar IMU data. For instance, shots that go straight through the net without touching or moving it much are often classified as misses, probably due to how it moves the net the same minute way a ball ricocheting off the back of the rim (which is a more common shot with more representative data points in the dataset) would.

Conclusion

Overall, the project demonstrates a promising application of IoT devices integrated with machine learning for real-time shot tracking. However there are still two main areas for improvement.

Firstly, the accuracy of inferences is still not high enough, and may classify shots wrongly. This undermines its utility as a tool for training feedback and improvement. Misclassified shots could mislead players about their true performance, making it challenging to accurately gauge and improve their skills. Ideally, the accuracy of inferences should be as close to 100% as possible, but this is challenging given due to the inherent variability in shot data and limitations of current machine learning models. Enhancing accuracy might require the integration of specialised sensors to determine shot outcomes deterministically based on pre-set thresholds.

Secondly, this project uses machine learning for the simple binary classification of miss or make, which only scratches the surface of potential insights that could be derived from detailed shot data. It does not fully leverage the capabilities of machine learning to classify the different shot patterns of the user. An improvement to this system could be to train models to categorise specific types of missed shots, such as those hitting the backboard, or particular sections of the rim. By expanding the data collected through additional sensors, the system could offer more detailed, actionable feedback, helping players understand and correct specific deficiencies in their shooting technique. However, this would likely require more complex models and by

extension more powerful hardware to support multi-class classification or even unsupervised learning algorithms, which are beyond the scope and hardware limitations of this project.

In conclusion, this project underscores the potential of integrating IoT and machine learning technologies in sports training. By enhancing the accuracy and expanding the analytical capabilities of the system, it could offer unprecedented precision and depth in training feedback, significantly enhancing players' performance and training experience.

References

1. Arduino. (n.d.). Datasheet ABX00027. Retrieved from https://docs.arduino.cc/resources/datasheets/ABX00027-datasheet.pdf

2. TensorFlow Blog. (2019, November). How to get started with machine learning. Retrieved from https://blog.tensorflow.org/2019/11/how-to-get-started-with-machine.html

3. Hackster.io. (n.d.). Basketball scoring table. Retrieved from https://www.hackster.io/508395/basketball-scoring-table-3b838c?f=1

4. Siegl, C. (n.d.). TensorFlow Lite for Microcontrollers adds support for efficient LSTM implementation. Medium. Retrieved from https://medium.com/@christoph-siegl/tensorflow-lite-for-microcontrollers-adds-support-for-efficient-lstm-implementation-25a5f7baa4f6

5. Netron. (n.d.). Netron. Retrieved from https://netron.app/

6. TensorFlow. (n.d.). Post-training quantization. Retrieved from https://www.tensorflow.org/lite/performance/post_training_quantization

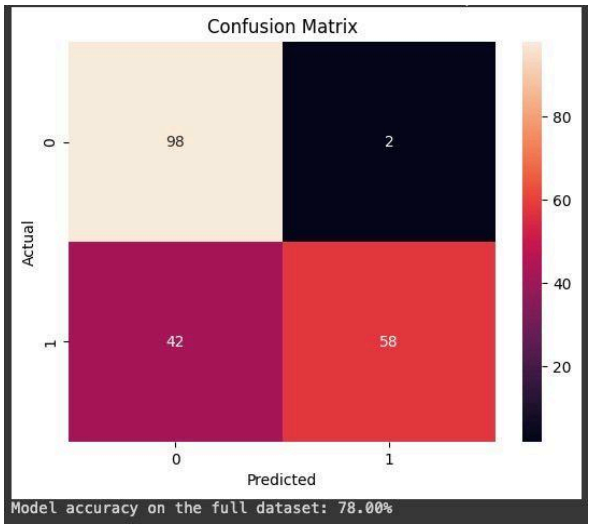
Appendix

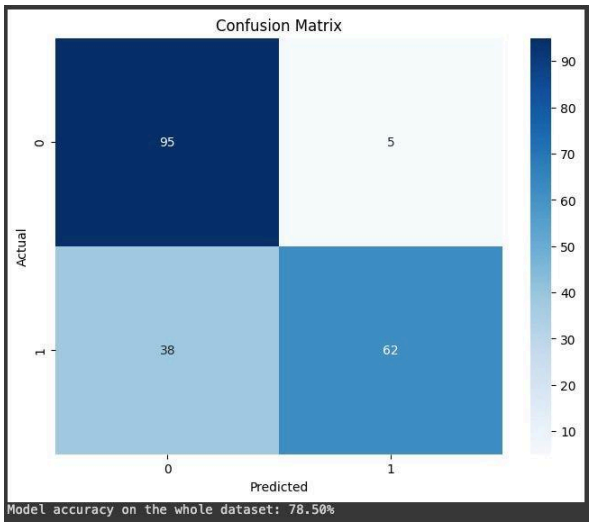
Colab Notebook:
https://colab.research.google.com/drive/11RBqhPx8TPL98d2DFzi0nclIiDHxptwpF#scrollTo=6qR9CBdoGZ74

MIT App Inventor code blocks used to develop app:

Demo Video:
https://youtu.be/IVYMZMagkUY

Confusion Matrices for LSTM Model (2 Layers) Before (left) and After (right) TFLite Conversion:





Confusion Matrices for Conv1D Model (1 Layer) Before (left) and After (right) TFLite Conversion: