NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CE4172 Tiny Machine Learning Project Report:

Chockalingam Kasi

April 5, 2023

Contents

Chapter 1

Introduction

1.1 Project Description

This project aims to demonstrate the potential of edge computing to perform gesture recogntion in real-time without relying on cloud-based processing. We use TensorFlow Lite to perform gesture recognition on an Arduino Nano 33 BLE device. Edge computing is a field that brings computation and data storage closer to the devices, rather than relying on a centralized data center or cloud. Edge computing reduces latency by performing computing tasks at the network's edge. TinyML is a subset of machine learning that deploys models on resource-constrained devices such as microcontrollers. TinyML models are typically small and optimized for low-power consumption, making them suitable for edge computing.

Our project demonstrates the benefits of TinyML and edge computing by using the Arduino Nano 33 BLE device to perform gesture recognition for the game StreetFighterIISpecialChampionEdition-Genesis. We train a Neural Network model to recognize twelve different gestures using data from the IMU sensors on the Arduino Nano 33 BLE. Edge computing allows players to execute actions in the game in real-time without relying on cloud-based computations.

1.2 StreetFighter II - Special Champion Edition

Figure 1.1: Street Fighter 2 game image

1.2.1 Game Description

Streetfighter 2 is a classic fighting game released by Capcom in 1991. The game features characters with unique fighting styles and special moves. The game works by engaging in one-on-one battles against each character in the game. The winner is determined by depleting the other player's health bar. Street Fighter 2 is widely recognized as the most popular game of all time and has various spinoffs based on the original games. The characters from the game, such as Ken, Ryu, and Chun-Li have become popular icons. This project attempts to play the game using human gestures triggering the movements.

1.2.2 Arcade Console keys and IMU custom gestures

Input	Arcade Console Key	Gesture Description
Left	←	Swing arm left
Right	→	Swing arm right
Up	↑	Jump
Down	↓	Crouch
Crouchpunch	crouch + punch	Crouch and punch
Hardpunch	→ + punch	Forward punch
Uppercut	→↓→ + punch	Upward punch
HighKick	→ + kick	High kick
Tatsumaki	↓←→ + kick	Elbow strike to the back
Hadoken	↓→→ + punch	Projectile attack
Crouch slide kick	↓→ + kick	Slide kick
Normal	Any button	Other motion

1.3 Data Collection

The data collection process involves reading data from an onboard accelerometer and gyroscope sensor using Arduino Nano 33 Ble. It uses the LSM9DS1 IMU sensor on the Arduino Nano 33 Ble board. It measures acceleration, angular velocity, and magnetic field strength in three dimensions. The data was collected by myself for the 12 gestures. The Arduino device waits to be triggered by waiting for a specific threshold of 2.5 Gs and above. Upon detecting significant motion, the code collects data for 119 samples. This reduces the amount of unnecessary data captured and captures the respective actions. A Python server is deployed to read data from the serial buffer and write the data to a CSV file for processing.

Arduino Data Collection properties

- Acceleration Threshold: 2.5

- NumSamples collected per sample: 119

1.4 Data Processing Steps

The sensor data for each gesture is combined and stored in a single Pandas data frame. For each recording, 119 consecutive samples of sensor data are taken and normalized to a range between 0 and 1. The data includes acceleration and gyroscope reading in the x, y, and z axes.

The input data and the respective labels are shuffled at random and split into three sets: training, validation, and testing in the ratio 70:15:15. These arrays are used to train, test and validate the tiny machine learning model, respectively. The resulting data consists of x_train, x_test, and x_validate containing the input data while the Y_train, Y_test, and Y_validate contain the respective one-hot encoding gesture labels. A total of 2240 actions were sampled. The data is split into 1568 Training data, 336 Validation data, and 336 Testing data.

Data Processing Steps

- Data is normalized between 0 to 1.

- Acceleration is changed to a range of -4 to 4.

- Gyroscope is kept to a range within -2000 to 2000.

- Normalized data is flattened into a single 714-dimensional vector.

- The corresponding one-hot encoded gesture label is stored separately.

Chapter 2

Implementation

2.1 Hardware

2.1.1 Hardware Setup

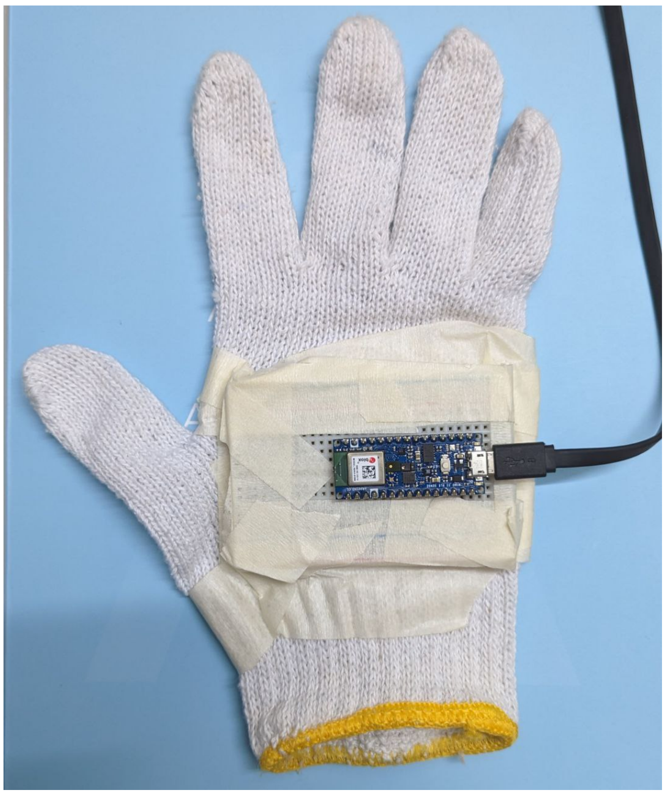

Figure 2.1: Arduino Nano attached to wearable gloves

2.2 Software

2.2.1 Tensorflow Lite Model

Model Architecture

Layer (type)	**Number of Neurons**	**Activation Function**	**Dropout Rate**
Dense	64	ReLU	NIL
Dense	64	ReLU	NIL
Dropout	64	NIL	0.1
Dense	32	ReLU	NIL
Dense	32	ReLU	NIL
Dropout	32	NIL	0.1
Dense	NUM_GESTURES	Softmax	NIL

Model Training Process

The model is a Sequential model built using the Keras API in TensorFlow. It contains 6 dense layers with ReLU activation functions and two dropout layers. The final dense layers have a Softmax activation function, which is used to classify the output into one of the possible gestures.

The model is compiled with a loss function and an optimizer. Mean squared error is used as the loss function and the Adam optimizer with a learning rate of 0.0001 is used. The model is trained with early stopping which is used to prevent overfitting of the model and thus lead to an improvement in generalization performance.

Early Stopping involves monitoring the performance of the model on a validation dataset during training and stopping the training process early if the performance on the validation dataset stops improving beyond the threshold. The early stopping process is very useful in ensuring that the model focuses on the patterns and relationships in the data. This helps the model to generalize better to new, unseen data.

Software optimizations

Dropouts are used in the TensorFlow lite model to prevent overfitting. The model uses two dropout layers, which is a regularization technique to prevent overfitting. The Dropout layers are randomly dropped out in the layer during the training process. This encourages the model to learn robust features that are not dependent on any single neuron, leading to better generalization performance.

Adam optimizer with a learning rate of 0.0001 is used for optimization. Adam optimizer is a popular optimization algorithm for neural networks as it combines both Adaptive Gradient Algorithm and Root Mean Square propagation optimization algorithms. It used adaptive learning rates and momentum to efficiently optimize the weights of the neural network.

Early Stopping is another form of regularization used to prevent overfitting. The earlyStopping callback monitors the validation loss during training and stops training if validation loss does not improve for a certain number of epochs. This helps to prevent overfitting the training data.

Quantization is a process of reducing the precision by representing them with a smaller number of bits. Quantization is a technique used to reduce memory size by converting high-precision weights and biases into lower-precision values. It helps to reduce the amount of computation needed for inference as the reduction in precision allows for faster computations. Quantized models tend to consume lesser energy due to lower calculations. The benefit of quantization is it allows models to be deployed on devices with limited storage space.

2.3 StreetFighter Gameplay

2.3.1 Overview

The Street Fighter Game is available in the OpenAI Gym Retro library. The open-source **retro** library contains a wide range of classic arcade games. It provides an environment that emulates the original version of the game console. The Gestures recognized by the Arduino Nano 33 Ble IMU sensor will be classified using the Tensorflowlite model and allow users to play the game in real time.

2.3.2 StreetFighter Gameplay Optimization

Threading is a tool that creates multiple threads within the program, with each thread performing different tasks concurrently. The two tasks that need to be performed concurrently would be running the OpenAI GymRetro Environment and the other thread running the Tensorflowlite Model to detect gestures. Threading is a technique that helps to improve the performance and robustness of the program by running multiple complex tasks to run concurrently.

- The `render_thread` and `gesture_thread` are created as instances of the `Thread` class from the `threading` module using the `target` parameter to specify the functions they will run in parallel:

  ```
  render_thread = threading.Thread(target=render_env)
  gesture_thread = threading.Thread(target=get_gesture_data)
  ```

- The `start()` method is called on both threads to start their execution:

  ```
  render_thread.start()
  gesture_thread.start()
  ```

- The `join()` method is called on both threads to wait for their completion before continuing with the main thread:

  ```
  render_thread.join()
  gesture_thread.join()
  ```

Chapter 3

Results

3.1 Quantized Model

3.1.1 Applying Quantized Model on Test Data

Gesture Name	**Correct**	**Incorrect**	**Total**	**Accuracy (%)**
left	20	2	22	90.91
right	24	0	24	100.00
jump	28	3	31	90.32
crouch	12	0	12	100.00
hardpunch	28	0	28	100.00
crouchpunch	17	1	18	94.44
crouchslide	51	2	53	96.23
highkick	18	1	19	94.74
uppercut	23	2	25	92.00
tatsumaki	25	1	26	96.15
hadoken	22	0	22	100.00
normal	51	5	56	91.07
Total	**319**	**17**	**336**	**94.94**

3.1.2 Quantization Model Analysis

Model	**Size (bytes)**
Basic Model	216812
Quantized Model	58016

Table 3.1: Comparison of model sizes before and after quantization

3.2 Summary

The project works towards building a gesture recognition model using neural networks and deep learning. The model was trained based on a dataset of gestures captured from an accelerometer and gyroscope sensors. The data was preprocessed to extract features and labels before training the model. The trained model was then optimized using quantization to reduce the model size and maintain performance accuracy. The quantized model performed as accurately as the original un-quantized model. The optimized model was evaluated on a test set and it achieved an overall accuracy of 94.94%.