Human Activity Tracker on the STM32 AI Platform

Solomiia Gadiichuk
Faculty of Applied Sciences
Ukrainian Catholic University
Lviv, Ukraine
hadiichuk.pn@ucu.edu.ua

Bohdan Zasymovych
Faculty of Applied Sciences
Ukrainian Catholic University
Lviv, Ukraine
zasymovych.pn@ucu.edu.ua

Marta Kosiv
Faculty of Applied Sciences
Ukrainian Catholic University
Lviv, Ukraine
kosiv.pn@ucu.edu.ua

Mentor : Yevgeniy Karplyuk
Kyiv, Ukraine
yevgeniy@karplyuk.kiev.ua

***Abstract*—This paper presents a human activity recognition (HAR) system developed for the STM32L475 IoT microcontroller board. The model was trained using the realworld dataset that contains accelerometer data. We are using a 1D Convolutional Neural Network (1D-CNN) to recognize activities. The model utilizes MaxPooling, GlobalAveragePooling, and post-training quantization to reduce the number of parameters and optimize the model for the STM32 microcontroller. To display data to the user, we use our own mobile application.**

***Index Terms*—Human Activity Recognition, STM32, Edge AI, Accelerometer, Gyroscope, Embedded Systems**

I. Introduction

The goal of our project is to design a compact and energy-efficient activity tracker that can recognize different types of human motion directly on an STM32 microcontroller without the need for external devices or cloud processing. The system uses acceleration and gyroscope data.

II. Experimental Setup

For this project, we use the STM32L475 IoT Discovery Kit (B-L475E-IOT01A1), a low-power microcontroller board made by STMicroelectronics. The board supports running small neural networks using the X-CUBE-AI tools, which allow machine learning models to be deployed directly on the microcontroller.

The most important components for our system are:

Motion sensor: The board includes an LSM6DSL 3-axis accelerometer and gyroscope, which we use to collect data about human movement and body orientation. These signals are later used for activity recognition.

Communication modules: The device has built-in Wi-Fi and Bluetooth 4.1 modules that can be used for data transfer in future versions of the project.

This setup allows us to record motion data directly from the board, process it locally, and prepare it for training and testing our activity detection model.

A. Power Management

To ensure the device is portable, we analyzed the power consumption. We connected an ammeter in series to the battery and measured a total current consumption of 70-80 mA. Using the special jumpers on the board, we also measured the current consumption of the microcontroller itself, which was approximately 10-20 mA.

Based on these measurements and the physical size of the board, we selected a Li-Pol battery with 3.7 V and 4000 mAh capacity. According to our calculations, this battery allows the tracker to operate autonomously for at least two days (48 hours).

B. Enclosure Design

We developed a custom 3D-printed case to attach the tracker to the user's upper arm. Since the development board is quite large, the case also has significant dimensions. However, the design was constructed to be as compact and comfortable as possible, providing secure protection for the electronics and the battery.

III. Dataset Description

In this work, we used the RealWorld dataset [3], which contains inertial sensor data collected from multiple body locations, including upper arm, wrist, and chest. The dataset includes accelerometer signals and other sensor readings recorded during everyday human activities.

The data were captured at a sampling frequency of 50 Hz, which is compatible with our STM32L475 IoT board. This allows direct usage of the dataset for training without resampling or frequency adjustment.

The dataset originally contains eight activity classes: walking, running, climbing stairs (up/down), jumping, lying, sitting, and standing. Each record contains linear acceleration values along the three axes (X, Y, Z). For this work, only accelerometer signals were used, although gyroscope and other sensor signals are available and can be incorporated in future work.

The total dataset size is approximately 18 hours of recordings. The dataset is well-documented, providing information about sensor placement and activity instructions, which helped us replicate a similar configuration on the STM32 board for real-time activity recognition.

A. Preprocessing

Before training, several preprocessing steps were applied to adapt the raw data for our model.

First, we modified the target classes to better suit our application. We combined lying, sitting, and standing into a single "rest" class, and removed the jumping class due to a significantly lower number of samples. This resulted in the final set of activity classes used for training.

To ensure consistent input distribution for the neural network, we applied Z-score normalization (standardization) to the accelerometer data. For each axis (x, y, z), we subtracted the mean value (μ) and divided by the standard deviation (σ). This transformation scales the data to have a mean of 0 and a standard deviation of 1, which helps in stabilizing the gradients during training and improves model convergence.

To prepare the data for training, we split the continuous recordings into overlapping windows of 96 samples (≈ 2 seconds) with a 75% overlap. This windowing strategy improves the model's ability to learn temporal dependencies and increases the total number of training samples.

In summary, the following steps were performed:

- Joining stationary classes (lying, sitting, standing) into one "rest" class and removing the "jumping" class.
- Data normalization by subtracting the mean and dividing by the standard deviation for each axis.
- Segmentation into windows of 96 samples with 75% overlap.
- Labeling each window with the correct activity ID.

The prepared dataset is compatible with real-time processing on embedded systems and suitable for training a lightweight neural network for human activity recognition on the STM32L475 microcontroller.

IV. Model Architecture

The Human Activity Recognition (HAR) task is done by 1D Convolutional Neural Network (1D-CNN) optimized for embedded systems. The model processes input windows of (96×3) samples (time steps $\times$ axes) to classify 5 activities, using $36,525$ trainable parameters.

A. Feature Extraction

The main part consist of three Conv1D layers for detecting local patterns:

- A MaxPooling1D layer (pool_size $= 2$) follows the first Conv1D (filters $= 32$), halving the sequence length from 96 to 46. This reduces computational load and memory usage.
- Two subsequent Conv1D layers (filters $= 64$ and filters $= 96$) extract progressively abstract features.
- Features are aggregated into a 96-dimensional vector using GlobalAveragePooling1D.

B. Classification Head

The dense head maps the feature vector to the final classification:

- The aggregated vector is stabilized with BatchNormalization.
- A hidden Dense(64) layer precedes the final Dense(7) layer.
- The final output utilizes a Softmax activation function to provide class probabilities.
- Dropout is used to prevent overfiting.

Memory Efficiency: The architecture is memory efficient and requires only $\mathbf{18.33}$ KiB of RAM out of $\mathbf{128}$ available for activations and runtime, ensuring feasibility for STM32 deployment. Weights are using $\mathbf{77.19}$ KiB of Falsh (ROM) out of $\mathbf{1024}$ available.

Inference Performance: The average inference time measured on the STM32L475 device is $\mathbf{44.22}$ ms. This latency is significantly lower than the data acquisition window, ensuring the system can process activities in real-time while leaving sufficient CPU resources for sensor management and BLE communication.

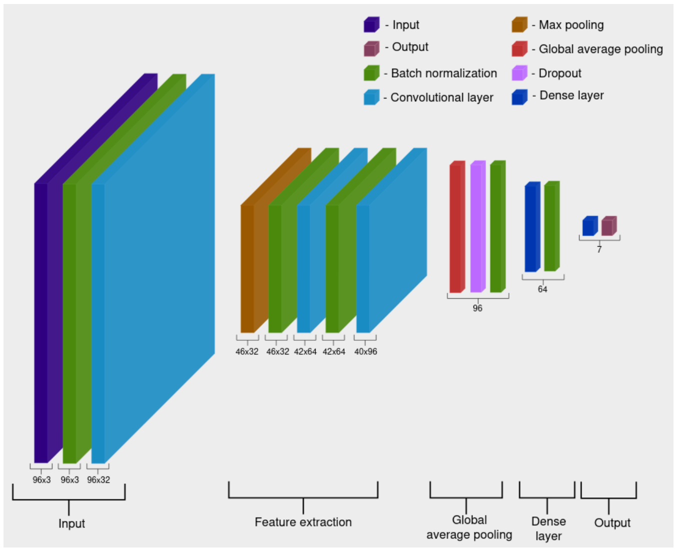

Fig. 1. Visual representation of the 1D-CNN Model Architecture, showing the flow from Input (96x3) through Conv1D, Pooling, and Dense layers.

V. Model Training

During training, the Categorical Cross-Entropy loss function was used to optimize parameter values.

Training stability and efficiency were maintained through the consistent use of Batch Normalization layers. To address the imbalance in the dataset, where certain activities (e.g., walking or resting) were overrepresented, we applied class weighting during training. This technique assigns higher penalties to misclassifications of minority classes, preventing the model from becoming biased toward the majority classes.

To further prevent overfitting, the training process employed Early Stopping, halting execution when the validation loss failed to decrease over several epochs.

Finally, to optimize the model for deployment on the STM32 microcontroller, we applied post-training quantization. This process converted the model weights and activations from 32-bit floating-point precision to 8-bit integers. This reduction significantly decreased the memory footprint and improved inference speed without a significant drop in accuracy.

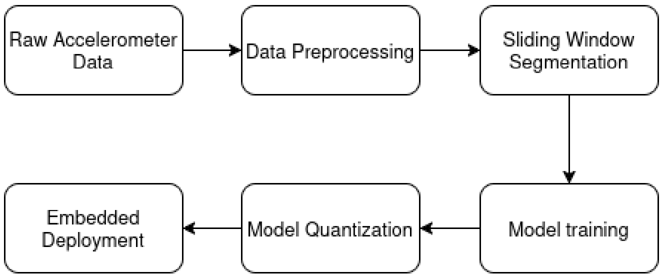

Fig. 2. Overview of the Model Training Pipeline, including data preprocessing, model compilation, training with class weights, and post-training quantization.

VI. Results

The performance of the trained 1D-CNN model was evaluated using a dedicated test set. Key metrics, including accuracy, precision, recall, and F1-score, were calculated, and a confusion matrix was generated.

A. Overall Performance Metrics

The model achieved an overall accuracy of **0.9422** (94.22%). A summary of the global performance metrics is provided below:

- **Accuracy:** 0.9422
- **Precision (Macro Avg):** 0.9484
- **Recall (Macro Avg):** 0.9318
- **F1-score (Macro Avg):** 0.9397

These metrics indicate a strong general performance across the different activity classes.

B. Confusion Matrix Analysis

The confusion matrix (Fig. 3), visually illustrates the model's classification performance for each of the 5 activity classes (labeled 0 through 4).

Key observations from the confusion matrix include:

- **Strong Diagonal:** The majority of predictions lie along the diagonal, indicating that the model largely correctly classifies each activity.
- **Inter-Class Confusion:** Some confusion happens between certain classes. For example, class 1 is often misclassified as class 4 (459 cases). Also, class 4 shows some confusion with class 1 (261 cases), likely due to the domination of class 4 in our dataset (over 11,000 samples).

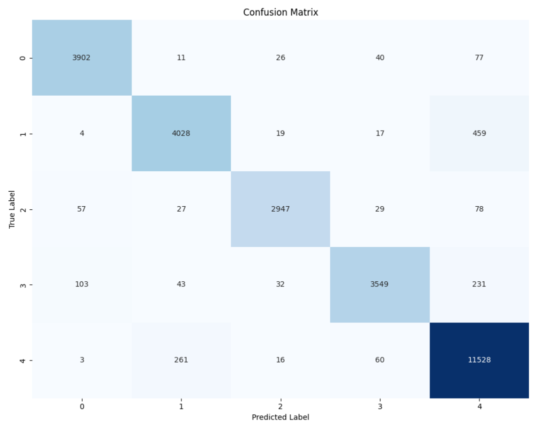

Fig. 3. Confusion Matrix of Model Predictions on the Test Set.

C. Detailed Classification Report

Breakdown of precision, recall, and F1-score for each individual class is provided in Table I.

From the table we can see that classes 0, 2, and 4 show high recall (> 0.93), while classes 1 and 3 have slightly lower recall values. Precision is consistently high across all classes, with class 2 achieving the highest value. Overall metrics show robust performance, though the macro average highlights slight disparities compared to the weighted average due to the significant class imbalance (Class 4 has significantly more samples).

TABLE I
Detailed Classification Report by Class

Class	Precision	Recall	F1-score	Support
0	0.9589	0.9620	0.9605	4056
1	0.9217	0.8898	0.9055	4527
2	0.9694	0.9391	0.9540	3138
3	0.9605	0.8967	0.9275	3958
4	0.9317	0.9713	0.9511	11868
Accuracy	**0.9422 (27547 total)**			
Macro Avg	0.9484	0.9318	0.9397	27547
Weighted Avg	0.9424	0.9422	0.9418	27547

VII. On-Device Inference Pipeline

Inference is performed fully on the STM32L475 microcontroller using the model generated by the X-CUBE-AI toolchain. The complete pipeline consists of sensor data acquisition, on-device normalization and quantization, neural network inference, output dequantization, and post-processing for prediction stabilization.

A. Input Preprocessing and Normalization

Raw accelerometer samples are acquired from the LSM6DSL sensor at a sampling frequency of 50 Hz. Each sample consists of three axes (x, y, z) represented as signed integers. Before being passed to the neural network, the data are converted to floating-point values and normalized using the same Z-score normalization parameters (μ, σ) that were computed on the training dataset:

$$x_{\text{norm}} = \frac{x - \mu}{\sigma}$$

The normalization constants are stored in flash memory and embedded directly into the firmware, ensuring consistency between training and inference.

B. Quantization of Input Data

Since the deployed model is fully quantized to 8-bit integers, the normalized floating-point inputs are quantized on-device. Each value is transformed using the input tensor scale (S_{in}) and zero-point (Z_{in}) provided by X-CUBE-AI:

$$x_q = \text{clip}\left(\left\lfloor \frac{x_{\text{norm}}}{S_{\text{in}}} \right\rceil + Z_{\text{in}}, -128, 127\right)$$

The quantized values are stored in the input activation buffer allocated by the AI runtime.

C. Output Dequantization

The network output is an 8-bit quantized tensor. To interpret the results, each output value is dequantized back to floating-point form using the output scale (S_{out}) and zero-point (Z_{out}):

$$y = (y_q - Z_{\text{out}}) \cdot S_{\text{out}}$$

The resulting vector represents the class probabilities after the Softmax layer.

D. Prediction Buffer and Temporal Filtering

To reduce sporadic misclassifications and sensor noise, a prediction buffer is used. The predicted class labels from the last N inference cycles are stored in a circular buffer. The final activity decision is obtained using majority voting over this buffer. This temporal filtering smooths rapid fluctuations between classes and improves perceived stability of the output, which is especially important for real-time user feedback.

The filtered activity label is then transmitted via BLE to the mobile application for visualization.

VIII. BLE Communication Setup

The STM32L475 IoT Discovery Kit includes an integrated SPBTLE-RF Bluetooth Low Energy (BLE) module, compliant with the Bluetooth 4.1 standard. This module is based on the ST BlueNRG-MS chipset and provides a complete BLE network processor solution.

The SPBTLE-RF module communicates with the STM32L475VGT6 microcontroller via the SPI3 interface, using dedicated IRQ and RESET control lines. The entire Bluetooth protocol stack is embedded inside the module firmware, which significantly reduces the processing load on the host MCU.

The main features of the SPBTLE-RF module include:

- Bluetooth V4.1 compliant, supporting multiple roles (sensor and hub) simultaneously.
- Integrated RF section with on-board chip antenna.
- Transmission power up to +4 dBm.
- Certified for CE, FCC, IC, and BQE compliance.
- On-field stack upgrading via SPI interface.

In our project, the BLE module was configured to transmit the recognized human activity label from the neural network running on the STM32 to a connected mobile device.

During implementation, the initial BLE example code provided by STMicroelectronics was adapted to support the B-L475E-IOT01A1 board configuration.

IX. Mobile Application

To visualize the results of the STM32 activity recognition, we developed a simple Android application using the Flutter framework. The goal of the app is to connect to the STM32 board through Bluetooth Low Energy and display the current activity class detected by the neural network. Based on the received activity data, the application performs post-processing, including local storage of activity records and visualization of historical activity information.

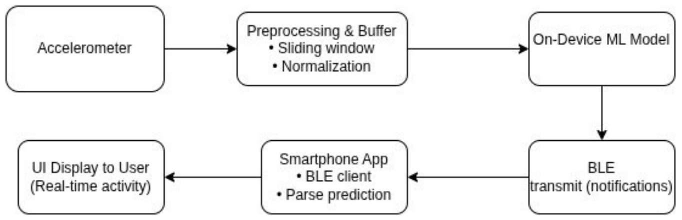

Fig. 4. Full Dataflow of the Application, showing the data path from the STM32 sensor to the mobile app interface and storage.

Acknowledgment

References

[1] STMicroelectronics, “How to perform motion sensing on STM32L4 IoTnode,” STM32 Wiki, 2024. [Online]. Available: https://wiki.st.com/stm32mcu/wiki/AI:How_to_perform_motion_sensing_on_STM32L4_IoTnode

[2] Edge AI Model Zoo, “Human Activity Recognition (HAR) model for STM32,” GitHub Repository, 2024. [Online]. Available: https://github.com/STMicroelectronics/stm32ai-modelzoo/tree/main/applications/HAR

[3] T. Sztyler and H. Stuckenschmidt, “On-body Localization of Wearable Devices: An Investigation of Position-Aware Activity Recognition,” *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2016. [Online]. Available: https://www.uni-mannheim.de/dws/research/projects/activity-recognition/dataset/dataset-realworld/