TinyML-Based Human Activity Recognition Using Sequence Models

P A Gautham Nair
ICT Academy of Kerala
Thiruvananthapuram, India
pagauthamnair@gmail.com

Abstract

Human Activity Recognition (HAR) is a critical application in wearable devices and IoT systems. In this study, I explore RNN, LSTM, and GRU models for HAR, optimizing them for TinyML deployment. The models are trained on a 12-class HAR dataset, converted into TensorFlow Lite format, and quantized for edge deployment. I evaluate their accuracy, inference time, and size, demonstrating real-time classification feasibility using mobile sensor data.

1 Introduction

Human Activity Recognition (HAR) is a rapidly growing field with applications in healthcare, fitness tracking, and smart environments. By analyzing motion sensor data, HAR enables intelligent systems to recognize and classify human activities such as walking, running, and sitting. Traditional deep learning models for HAR require significant computational resources, making them unsuitable for deployment on low-power edge devices.

With the advent of TinyML, machine learning models can be optimized for microcontrollers and embedded systems with constrained memory and processing power. However, deploying sequence models such as RNN, LSTM, and GRU on TinyML poses challenges in terms of model size, inference time, and energy efficiency.

This study aims to explore the feasibility of deploying HAR models on edge devices by:

- Training and optimizing RNN, LSTM, and GRU models on sequential sensor data.
- Converting models to TensorFlow Lite and applying quantization to reduce memory footprint.
- Evaluating models based on accuracy, inference time, and model size.
- Implementing real-time motion classification using a mobile phone's accelerometer and gyroscope.

My contributions include an end-to-end HAR system optimized for TinyML deployment, along with a detailed comparison of full, TFLite, and quantized models. The findings of this study provide insights into the trade-offs between accuracy, speed, and model size in resource-constrained environments.

2 Related Work

Human Activity Recognition (HAR) has been extensively studied, with various approaches that use deep learning models for accurate classification. Traditional methods relied on hand-crafted feature extraction combined with classical machine learning algorithms such as Support Vector Machines (SVM) and Random Forests [2]. However, these approaches struggled to generalize across diverse activity patterns.

Recent advances in deep learning have led to the adoption of sequence models such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU) for HAR. For instance, Ordóñez and Roggen [5] demonstrated that deep LSTM networks outperformed conventional classifiers on wearable sensor data, achieving high recognition accuracy across multiple activity classes. Similarly, Hammerla et al. [3] compared various deep learning architectures, highlighting the superior performance of LSTM LSTM in modeling temporal dependencies.

Despite the effectiveness of deep learning models, deploying them on resource-constrained edge devices remains a challenge. Research on TinyML focuses on optimizing neural networks for low-power environments, primarily through model compression techniques like quantization and pruning. Lane et al. [4] explored model compression techniques for mobile deep learning, demonstrating that quantization could significantly reduce model size while maintaining accuracy. Furthermore, Bannur et al. [1] proposed a TinyML-based HAR model using TFLite, showing that downsized models could achieve real-time inference with minimal energy consumption.

Our work builds upon these advancements by systematically comparing RNN, LSTM, and GRU models in a TinyML setting. I evaluate the trade-offs between accuracy, inference time, and memory footprint, extending previous research by implementing real-time activity recognition using mobile sensor data.

3 Methodology

Our approach to TinyML-based Human Activity Recognition (HAR) involves data collection, preprocessing, model selection, optimization, and deployment. This section details each step in our methodology.

3.1 Dataset

I have used the Smartphone-Based Recognition of Human Activities and Postural Transitions dataset [6], which contains motion sensor data from 30 participants aged 19 to 48 years. The dataset was collected and curated by Jorge Luis Reyes-Ortiz, Luca Oneto, Alessandro Samà, Xavier Parra, and Davide Anguita. The participants performed six basic activities—standing, sitting, lying, walking, walking upstairs, and walking downstairs—along with postural transitions such as sit-to-stand and stand-to-lie.

The dataset was collected using a Samsung Galaxy S II smartphone worn on the waist, capturing 3-axial accelerometer and gyroscope readings at a sampling rate of 50Hz. The raw sensor signals were processed using noise filtering and segmented into fixed-width sliding windows of 2.56 seconds with a 50% overlap. Each window

contains 128 readings and is represented as a 561-feature vector derived from time and frequency domain computations.

3.2 Preprocessing

The raw data was provided in individual text files for each participant and sensor type, following the format `experimentXX_personYY.txt`. This resulted in a total of 60 individual files—30 for accelerometer data and 30 for gyroscope data. Additionally, metadata about each file was stored in a separate file named `activity_labels.txt`. The preprocessing workflow involved the following steps:

- Combining all 60 raw data files based on their respective experiment and participant identifiers to create a unified dataset.
- Saving the merged data in a structured CSV format for further processing.
- Identifying class imbalance in the dataset.
- Applying Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalance. Specifically, under-represented classes are oversampled and over-represented classes are undersampled.
- Balancing the dataset to a final size of 45,000 samples, ensuring equal distribution across all activity classes.

These preprocessing steps ensured that the dataset was well-structured and balanced before training machine learning models.

3.3 Model Selection

To effectively capture temporal dependencies in human motion patterns, I experimented with three sequence models: Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU). These models were chosen for their ability to process sequential data, making them well-suited for time-series classification tasks such as human activity recognition.

The primary objective was to compare the performance of these models in terms of classification accuracy, inference time, and model size, particularly in the context of TinyML deployment. RNNs provide a foundational approach to sequential modeling, but they suffer from vanishing gradient issues over long sequences. LSTMs and GRUs, on the other hand, introduce gating mechanisms to mitigate these issues and enhance long-term dependency learning. By training and evaluating all three models, I aimed to analyze their strengths and limitations in a resource-constrained environment and determine the most suitable model for real-time activity recognition.

3.4 Optimization for TinyML

Deploying deep learning models on resource-constrained edge devices requires reducing memory usage and computational complexity while maintaining accuracy. To achieve this, I optimized the trained RNN, LSTM, and GRU models using TensorFlow Lite (TFLite).

3.4.1 Model Conversion and Quantization. The full-precision models were converted to TFLite format using the TensorFlow Lite Converter. To further reduce the model size and improve efficiency, I applied TFLite's built-in quantization method. This process lowers numerical precision while preserving model functionality, making it feasible to deploy the models on low-power embedded systems.

3.4.2 Deployment Readiness. The quantized models significantly reduce memory footprint and computational demands, allowing real-time inference on edge devices. These optimizations ensure that the TinyML-based human activity recognition system can operate efficiently with minimal hardware resources.

3.5 Deployment and Real-Time Evaluation

To enable real-time motion recognition, I developed a streaming pipeline that captures sensor data from a mobile device and processes it for activity classification.

3.5.1 Sensor Data Acquisition. For data collection, I utilized the open-source **Sensor Server** application [7], which streams real-time accelerometer and gyroscope data from a smartphone. The Sensor Server application, licensed under the GNU General Public License v3.0 (GPL-3.0), operates as a WebSocket server on the mobile device, transmitting sensor readings over a network connection.

3.5.2 Backend Processing. A FastAPI backend was implemented as a WebSocket client to establish a connection with the Sensor Server and receive the incoming sensor data stream. The backend handled preprocessing steps such as normalization and data formatting before passing the processed data to the optimized TinyML models for real-time activity classification.

3.5.3 Frontend Visualization. To display predictions in real time, I developed a simple local webpage that connected to the FastAPI backend. The webpage dynamically updated the recognized activity as new predictions were generated, providing an interactive visualization of motion classification results.

This methodology and deployment pipeline enables efficient and responsive human activity recognition, leveraging real-time sensor data while maintaining computational efficiency for TinyML applications. or real-time visualization.

4 Experiments and Results

This section presents the experimental setup, training performance, evaluation metrics, and a comparative analysis of the models in terms of accuracy, inference time, and model size.

4.1 Experimental Setup

The models were trained and evaluated using Python with TensorFlow. The learning rate was periodically adjusted using the `ReduceLROnPlateau` scheduler to enhance convergence. The following hyperparameters were used for all models:

- Optimizer: Adam
- Learning rate: ReduceLROnPlateau (factor=0.5)
- Batch size: 32
- Number of epochs: 50

The dataset was split into training and testing sets in an 80:20 ratio. Model performance was evaluated using accuracy, precision, recall, and F1-score.

4.2 Training Performance

To evaluate the learning behavior of the models, we trained RNN, LSTM, and GRU architectures for 50 epochs using a batch size of 32.

Figure 1 shows the training accuracy of each model across epochs, while Figure 2 illustrates the corresponding training loss curves.

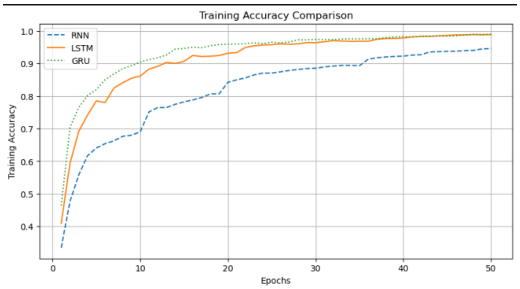

Figure 1: Training accuracy comparison of RNN, LSTM, and GRU over 50 epochs.

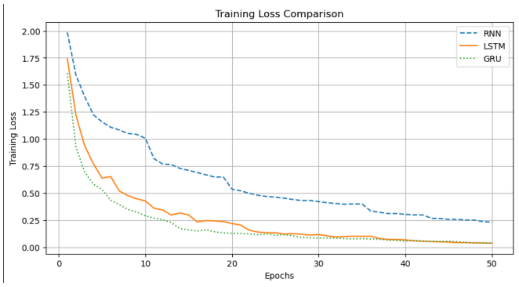

Figure 2: Training loss comparison of RNN, LSTM, and GRU over 50 epochs.

4.3 Evaluation Metrics

The classification performance of each model was evaluated using accuracy, precision, recall, and F1-score. The results are summarized in Table 1.

Table 1: Evaluation Metrics for RNN, LSTM, and GRU Models

Model	Accuracy	Precision	Recall	F1-score
RNN	0.9446	0.9478	0.9446	0.9424
LSTM	0.9835	0.9839	0.9835	0.9834
GRU	0.9850	0.9853	0.9850	0.9849

4.4 Model Comparison

Figures 3, 4, and 5 illustrate the differences between the full, TFLite, and quantized versions of the RNN, LSTM, and GRU models in terms of accuracy, inference time, and model size.

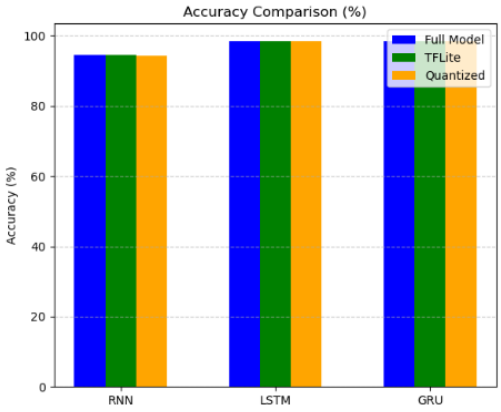

Figure 3: Comparison of model accuracy for full, TFLite, and quantized versions.

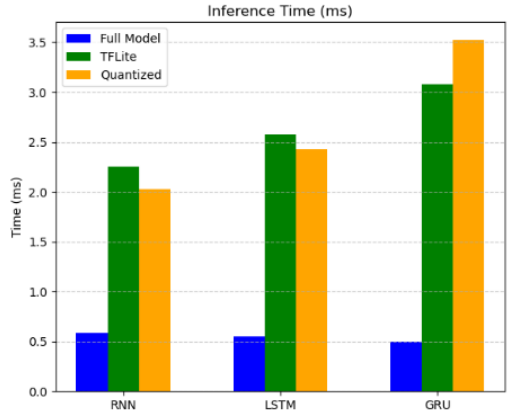

Figure 4: Comparison of inference time for full, TFLite, and quantized versions.

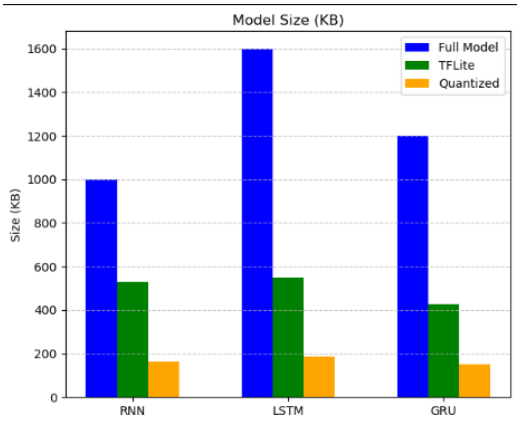

Figure 5: Comparison of model size for full, TFLite, and quantized versions.

The analysis highlights that GRU is the most efficient model for real-time applications, as it provides high accuracy while maintaining a smaller size and faster inference speed than LSTM. The RNN, while computationally lighter, does not achieve comparable accuracy, making it less suitable for precise activity recognition tasks.

4.5 Real-Time Prediction with Quantized Models

After evaluating the models on the test dataset, the quantized versions were deployed for real-time motion recognition. Sensor data was continuously collected from a mobile device and transmitted to the FastAPI backend in real time. The data was processed and fed into the quantized models in batches of 50. Each model generated a predicted activity label for every batch, which was then displayed on the local webpage.

The models successfully maintained continuous predictions with minimal latency, demonstrating their effectiveness for real-time deployment on edge devices.

5 Conclusion and Future Work

5.1 Conclusion

This study successfully developed and optimized RNN, LSTM, and GRU models for human activity recognition on mobile sensor data. The models were compared based on accuracy, inference time, and model size. The results indicate that:

- The LSTM and GRU models significantly outperformed the RNN in terms of accuracy, demonstrating their superior ability to capture temporal dependencies.
- GRU achieved the highest accuracy (98.50%) while maintaining a lower inference time than LSTM, making it the best balance of speed and performance.
- Quantization effectively reduced model size and inference time without major accuracy loss, proving beneficial for TinyML deployment.

Overall, the GRU model emerges as the most balanced choice, offering high accuracy and fast inference, while the LSTM model remains a strong alternative for slightly higher accuracy at the cost of increased computational demand.

Real-time evaluation confirmed that the quantized models could successfully process incoming sensor data in batches of 50 and make continuous activity predictions displayed on a local webpage.

5.2 Limitations and Challenges

One major limitation observed was that the models tended to fixate on certain activity labels, particularly “walking_downstairs” and “lie_to_stand.” This is likely due to differences in preprocessing—while the training data underwent noise and gravity removal using a Butterworth low-pass filter, the real-time data did not. This discrepancy may have caused inconsistencies in model predictions. Addressing this issue requires applying the same preprocessing techniques to the real-time data before feeding it into the models.

5.3 Future Work

The next step is to deploy this system on the internet, making it accessible to a wider audience. The planned deployment consists of the following steps:

- Developing a web interface where users can transmit mobile sensor data.
- Hosting the FastAPI backend on Render to handle real-time data processing.
- Implementing a frontend using Streamlit to visualize real-time activity predictions.

A key challenge in this deployment is ensuring smooth and reliable communication between the webpage and the FastAPI backend while maintaining real-time responsiveness. Future work will focus on addressing these challenges and optimizing the system for real-world applications.

References

[1] Bhavya Bannur, Ajay Kumar Sharma, and Ravi Vijayakumar. 2021. TinyHAR: TinyML-Based Human Activity Recognition Using Low Power Edge Devices. *IEEE Sensors Journal* 21, 2 (2021), 1645–1653.

[2] Andreas Bulling, Ulf Blanke, and Bernt Schiele. 2014. A Tutorial on Human Activity Recognition Using Body-worn Inertial Sensors. *ACM Computing Surveys (CSUR)* 46, 3 (2014), 1–33.

[3] Nils Y. Hammerla, Shane Halloran, and Thomas Ploetz. 2016. Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables. In *Proceedings of IJCAI*. 1533–1540.

[4] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Christos Mascolo, Cecilia Mascolo, and Fahim Kawsar. 2017. Squeezing Deep Learning into Mobile and Embedded Devices. *IEEE Pervasive Computing* 16, 3 (2017), 82–88.

[5] Felix Ordóñez and Daniel Roggen. 2016. Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. *Sensors* 16, 1 (2016), 115.

[6] Jorge L. Reyes-Ortiz, Luca Oneto, Albert Samà, Xavier Parra, and Davide Anguita. 2015. Transition-Aware Human Activity Recognition Using Smartphones. *Neurocomputing* (2015).

[7] Umer0586. 2023. Sensor Server: Stream mobile sensor data over WebSockets. https://github.com/umer0586/SensorServer GNU General Public License v3.0.