Project Report: Efficient Activity Recognition on Memory-Constrained Smart Watches

Emin Abdurahmanov, Saju Khakurel, Thong Phan
Supervised by Prof. Kristof Van Laerhoven, University of Siegen,
18 July 2025

***Abstract*—This project evaluates the implementation of deep learning models for Human Activity Recognition (HAR) on the memory-constrained Bangle.js 2 smartwatch. The study utilizes three distinct datasets: HANG Time-HAR (basketball activities), PAMAP2 (common physical activities), and WEAR (outdoor fitness activities). Four main architectures—LSTM, CNN, a Hybrid CNN-LSTM, and DeepConvLSTM—are analyzed, along with a SeparableConv1D model designed for edge devices. Post-training 8-bit integer quantization using TensorFlow Lite for Microcontrollers (TFLM) was applied to reduce model size and meet hardware constraints.**

The results vary by dataset. The DeepConvLSTM model yielded the highest test accuracies of 79.3% on HANG Time-HAR and 90.5% on PAMAP2. For the PAMAP2 dataset, the standard CNN was also evaluated, achieving 85.7% accuracy with lower architectural complexity. On the WEAR dataset, the SeparableConv1D architecture achieved the highest F1-score (69.00% on merged classes) while requiring less memory and lower inference latency than the baseline CNN in that study. The findings indicate that model performance and suitability for deployment are dependent on the specific dataset, activity complexity, and hardware resource limitations.

***Index Terms*—Human Activity Recognition, Smart Watch, CNN, LSTM, CNN-LSTM, TinyML, Quantization, Edge AI, Confusion Matrix**

I. Introduction

The ubiquity of sensor-equipped smartwatches presents a significant opportunity for real-time Human Activity Recognition (HAR), enabling applications from automated fitness tracking to context-aware personal assistants. While deep learning models, particularly Convolutional Neural Networks (CNNs), have demonstrated state-of-the-art performance in HAR, their high computational and memory requirements often render them unsuitable for deployment on resource-constrained wearable devices. The field of TinyML aims to bridge this gap by developing highly efficient models that can run locally on-device, preserving user privacy and ensuring low-latency inference without relying on cloud connectivity.

This project discovers the application of on-device HAR using the Bangle.js 2 smartwatch, an open-source platform equipped with multiple sensors, including a 3-axis accelerometer. The primary technical challenge lies in deploying computationally intensive models on constrained hardware, which features only 256KB of RAM and 1024KB of on-chip Flash memory [5]. These limitations make standard DL frameworks infeasible. To overcome this constraint, our approach leverages the TensorFlow Lite for Microcontrollers (TFLM) framework, which is designed to execute highly optimized neural networks that are memory-efficient, exhibit low latency, and maintain performance comparable to their full-sized counterparts [8].

II. Case Study 1: HANG Time-HAR

This study aims to recognize basketball players' activities using Human Activity Recognition (HAR) techniques on the Bangle.js 2 smartwatch. This device records the tri-axial accelerometer data at a sampling rate of 50 Hz with an acceleration range ±8g. Using the HANG Time-HAR dataset [1], which was collected from 24 players across North America and Europe, the objective is to classify 19 basketball-related activities like dribbling, running, or shoots. We used some lightweight deep learning models such as Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNN), Hybrid CNN-LSTM, and DeepConvLSTM, which were developed and optimized for real-time, low memory, and deployment using TensorFlow Lite.

A. Dataset Overview

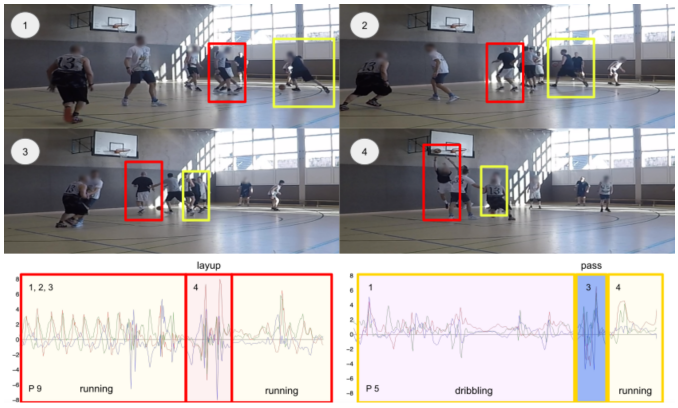

Fig. 1: A scene and activities from the dataset: Offense play of player 12 (yellow) and player 6 (red), with player 12 dribbling the ball (1), (2), and then passing (3) it to player 6. Player 6 then performs a layup (4). Video frames 1-4 and the performed activities are highlighted in the time series below. [1]

This dataset is a benchmark for evaluating physical human activity recognition (HAR) using wrist-worn inertial sensors,

which provide 3D data that is then tailored for basketball training, drills, and games. The dynamic movements in basketball make it ideal for wrist-based monitoring, resulting in applications such as in game analysis, guided training, and personal tracking.

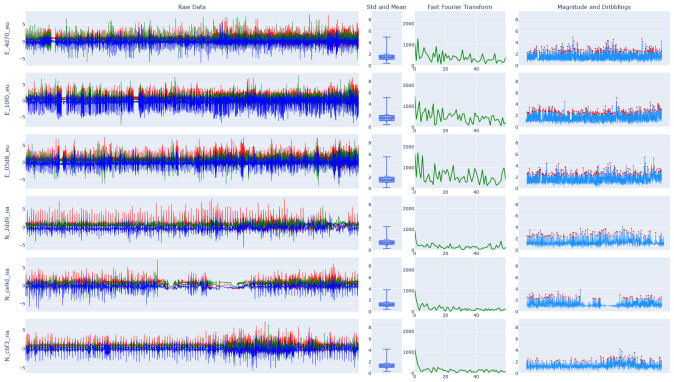

Fig. 2: Feature analysis of the class dribbling for players 4d70_eu, 10f0_eu, and 05d8_eu (experts) and 2dd9_na, ce9d_na, and c6f3_na (novices). The plot consists of 4 columns: (1) Raw data as recorded during the dedicated dribbling drill (approx. 7 min of data (Germany) and 5 min of data (USA)). The X-axis is in red, the Y-axis in green, and the Z-axis in blue. (2) Standard deviation (diamond shape), median, interquartile q_1 and q_3 (rectangle shape), and upper and lower fences. (3) Fast Fourier Transformation. (4) Local maxima (prominence = 1.4) using the magnitude of the input signal (1), with each red dot indicating a dribbling peak. [1]

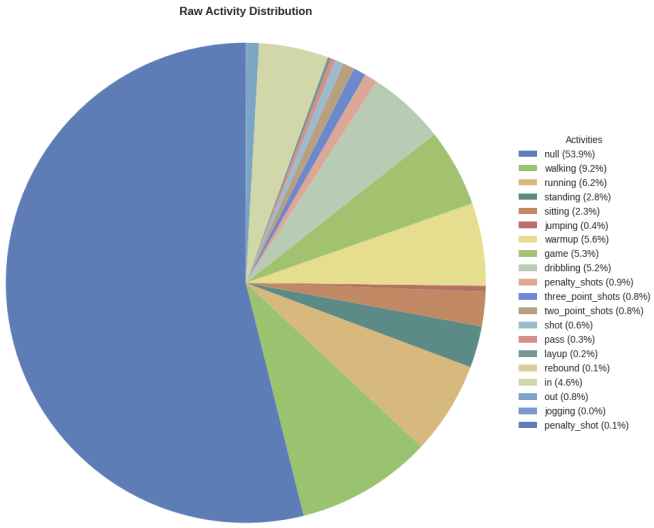

Fig. 3: A breakdown of the 19 activity classes in the HANG TIME-HAR dataset with distribution.

The dataset was collected from 24 players across two teams during a few structured training sessions and matches. One team was from the United States and the other from Germany. Data collection was conducted in training facilities and game venues in both countries, capturing the cultural differences in basketball rules and playing styles. It also captures the range of participant skill levels from novices to experienced players. Each player's data was stored in a CSV file with the 3-axis accelerometer data, movements defined in four sectors: coarse, basketball, locomotion, in/out, and the respective timestamp.

B. Pre-processing Techniques

We pre-processed the raw accelerometer data collected from the wrist-worn sensor, in this case the bangle, during basketball training, drills, and games. The data, stored in CSV files, was initially loaded as data type, float32 for the acceleration axes (acc_x, acc_y, acc_z) and activity labels (locomotion, coarse, basketball, in/out) respectively. The initial step was data cleaning that involved removing rows with missing acceleration values and filtering out extreme measurements ($|acc_x|$, $|acc_y|$, $|acc_z| > 8$) to ensure data quality. Then, to optimize the memory usage, the data was downsampled from an original 50 Hz to 10 Hz by selecting every 5th sample, resulting in reduced data volume while preserving key temporal patterns.

Secondly, the pre-processed data was segmented into 2-second sliding windows, equivalent to 20 samples at 10 Hz, with a 50% overlap (1-second step size) to capture the dynamic basketball movements effectively. These windows were labeled based on the predominant activity within each segment, extracted incrementally from each player's CSV file using parallel processing. To manage memory constraints, a maximum limit of 5,000 sequences per activity was enforced using random sampling. After that, the data was split into training, validation, and test sets: 60% for training, 20% for validation, and 20% for testing. The acceleration data within each split was normalized using MinMaxScaler to ensure consistent scaling across the (20, 3) input shape. Individual activity datasets were then combined by concatenating all sequences and assigning integer mapping, resulting in a total

TABLE I: Dataset Summary: Activity Sequences and Durations.

Activity	Total	Train	Val.	Test	Duration (s)
null	5000	3000	1000	1000	10000.0
running	5000	3000	1000	1000	10000.0
walking	5000	3000	1000	1000	10000.0
standing	5000	3000	1000	1000	10000.0
penalty shots	5000	3000	1000	1000	10000.0
sitting	5000	3000	1000	1000	10000.0
warmup	5000	3000	1000	1000	10000.0
dribbling	5000	3000	1000	1000	10000.0
in	5000	3000	1000	1000	10000.0
game	5000	3000	1000	1000	10000.0
three point shots	4753	2851	951	951	9506.0
out	4501	2700	900	901	9002.0
two point shots	4420	2652	884	884	8840.0
shot	3336	2001	667	668	6672.0
jumping	2216	1329	443	444	4432.0
pass	1727	1036	345	346	3454.0
layup	1250	750	250	250	2500.0
rebound	374	224	75	75	748.0
penalty shot	287	172	57	58	574.0
jogging	174	104	35	35	348.0

of 43,819 training sequences, 14,607 validation sequences, and 14,612 test sequences across 19 classes, as detailed in the master dataset output. The resulting dataset preparation is summarized in Table I, which shows the number of sequences allocated to each set for each activity class. The table highlights an imbalanced distribution, with activities such as walking, running, and standing contributing the maximum of 5,000 sequences. But the more rare activities like jogging (174 sequences) and rebounding (374 sequences) the distribution is less. This imbalance is further visualized in Figure 4, where the bar chart illustrates the varying prevalence of each class, with custom colors for enhanced visual distinction.

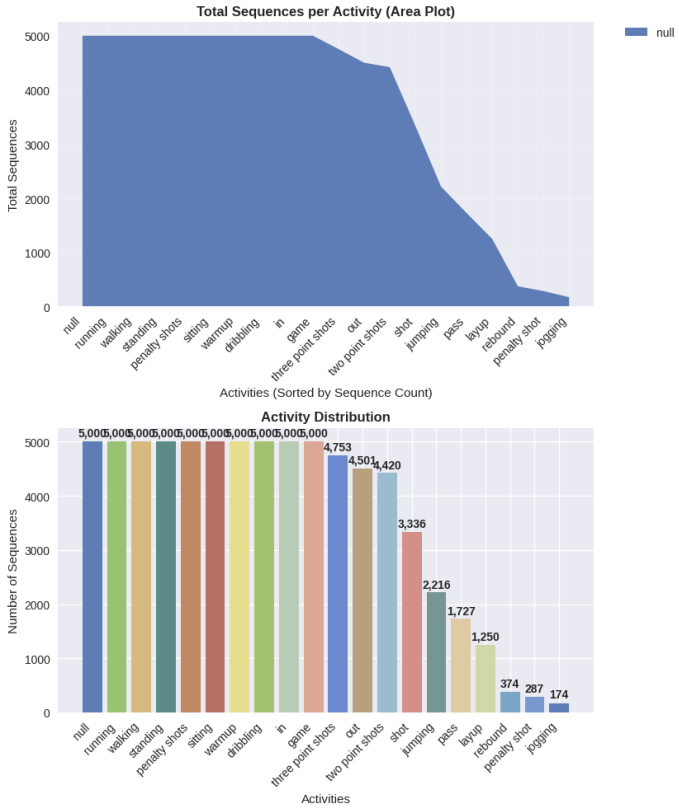

Fig. 4: Distribution of Activity Sequences Across Classes

Note: The figure includes an area plot (top) showing the cumulative total sequences per activity and a bar chart (bottom) with sequence counts, highlighting the imbalanced distribution. Numbers on bars indicate exact counts, and colors are assigned using a custom palette for visual distinction.

C. Model Architecture

This study evaluates four distinct neural network architectures tailored for time-series classification of the sensor data, which is then optimized for resource-constrained environments such as TinyML applications. Each model uses the sequential processing of 1D input data (20 timesteps, 3 channels) derived from a master multiclass dataset. The architectures are designed to balance performance and computational efficiency, with the trainable parameters.

1) LSTM: The Long Short-Term Memory (LSTM) model, as illustrated in Figure 5, is a recurrent neural network architecture. It is specifically designed to detect complex temporal dependencies in sequential data. This type of model is well-suited for handling long-term patterns in correlated time-series data, making it a robust choice for multi-class sensor data classification. The model processes 1D input data with a shape of (20 timesteps, 3 channels). Its comprehensive design results in a total of 8,524 trainable parameters.

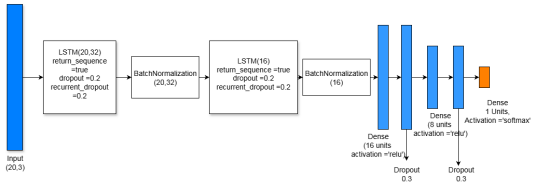

Fig. 5: LSTM Architecture Diagram

The detailed layer-by-layer structure of the LSTM model is presented in Figure 6. The architecture starts with an input layer that receives sequences of data in shape (20, 3). Following this, the model employs two stacked LSTM layers. The first LSTM layer consists of 32 units. It utilizes a dropout rate of 0.2 and is configured to return the full sequence of its hidden states to preserve temporal context. It is then immediately followed by a batch normalization layer, which is incorporated to stabilize and accelerate the training process. The second LSTM layer contains 16 units, maintaining the same dropout configuration as the first. However, this layer is set to output a fixed-length representation (i.e., the hidden state of the last time step), effectively summarizing the learned temporal features into a concise vector. This is again followed by a Batch Normalization layer, further enhancing training stability.

Subsequently, the model transitions to fully connected Dense layers. There are two such layers: the first with 16 units and the second with 8 units. Both dense layers utilize the Rectified Linear Unit (ReLU) as their activation function for introducing non-linearity. To improve generalization and reduce overfitting, each Dense layer is immediately followed by a Dropout layer, with rates of 0.3 and 0.2, respectively. The final output layer is a Dense layer, with the number of units corresponding to the total number of activity classes. This layer employs a Softmax activation function to produce normalized class probabilities, suitable for multi-class classification.

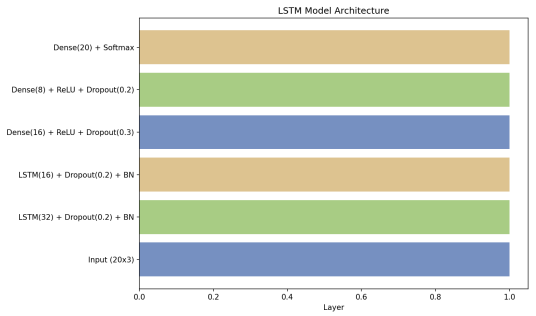

Fig. 6: LSTM Model Layer Representation (Bar Chart)

2) CNN: The Convolutional Neural Network (CNN) model employs convolutional layers (16, 32, and 16 filters) with pooling and dense layers to extract spatial features from 1D data, resulting in 4,124 trainable parameters. This type of architecture thrives in identifying local patterns from sensor inputs.7

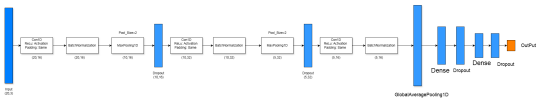

Fig. 7: CNN Architecture Diagram showing convolutional layers for feature extraction and dense layers for classification.

The detailed layer configuration of the CNN model is presented in the bar-graph 8. The architecture begins with an input layer of shape (20, 3). This is then followed by three 1D Convolutional (Conv1D) blocks. The first block uses 16 filters, ReLU activation, Batch Normalization, and Max Pooling (pool size 2) with a dropout rate of 0.2. The second block employs 32 filters, ReLU activation, Batch Normalization, and Max Pooling (pool size 2) with a 0.2 dropout rate. The third Conv1D block utilizes 16 filters, ReLU activation, and Batch Normalization, but instead of Max Pooling, it connects to a Global Average Pooling layer. This global pooling layer effectively reduces each feature map to a single value, preparing the data for the fully connected layers. Following the convolutional and pooling layers, the model transitions to two fully connected (Dense) layers. The first Dense layer has 16 units, followed by ReLU activation and a Dropout layer with a 0.3 rate. The second Dense layer has 8 units, also with ReLU activation and a 0.3 dropout rate. The final output layer is a Dense layer with 20 units and a Softmax activation function, producing class probabilities for multi-class classification.

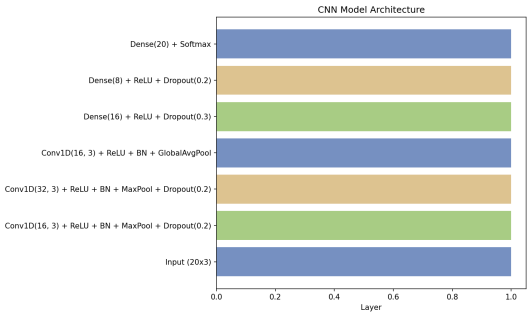

Fig. 8: CNN Layer Representation (Bar Chart)

3) Hybrid: The Hybrid model 9 combines the strengths of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. This architecture integrates spatial feature extraction with temporal modeling, providing a balanced approach for enhanced performance on complex time-series tasks from multi-class sensor data. It processes 1D input data of shape (20 timesteps, 3 channels) and comprises a total of 4,476 trainable parameters.

Fig. 9: Hybrid Architecture Diagram combining convolutional and LSTM layers for feature extraction and temporal modeling.

The detailed layer composition of the Hybrid model is shown in bar-graph 10. The model begins with an input layer receiving sequences of shape (20, 3). The convolutional branch consists of two 1D Convolutional (Conv1D) layers, both with 16 filters and ReLU activation. The first Conv1D layer is followed by Batch Normalization (BN). The second Conv1D layer is also followed by BN, then Max Pooling (pool size 2), and a Dropout layer with a 0.2 rate. These layers are designed to extract local spatial features from the input sequences.

Following the convolutional layers, the model integrates a recurrent branch with two LSTM layers. The first LSTM layer has 16 units, a dropout rate of 0.2, and is configured to return its full sequence of hidden states. The second LSTM layer has 8 units, a dropout rate of 0.2, and a recurrent dropout of 0.2. This LSTM layer is followed by a Batch Normalization layer. It should be noted that for the subsequent Dense layers to process the output, a Flatten layer (not explicitly shown but implied) would typically be used after the last Batch Normalization to convert the sequence output into a single vector, or TimeDistributed Dense layers would be utilized.

Finally, the model includes two fully connected (Dense) layers. The first Dense layer has 16 units, followed by ReLU activation and a Dropout layer with a 0.3 rate. The second Dense layer has 8 units, also with ReLU activation and a 0.2 dropout rate. The ultimate output layer is a Dense layer with 20 units and a Softmax activation function, generating normalized class probabilities for multi-class classification.

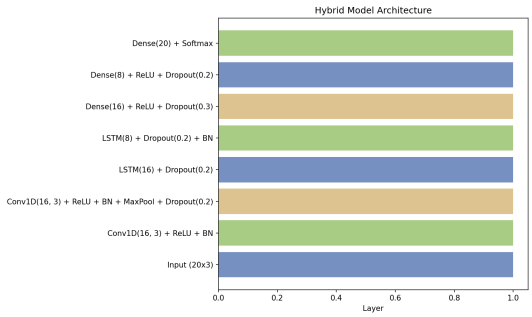

Fig. 10: Hybrid Model Layer Representation (Bar Chart).

4) DeepConvLSTM: The DeepConvLSTM model 11 features a deeper architecture with two convolutional blocks (32 and 64 filters) for robust feature extraction, which is then followed by LSTM layers (32 and 16 units) and again by dense layers, giving 41,324 trainable parameters. Since it is designed with TinyML constraints, it prioritizes accuracy on challenging multi-class datasets.

Fig. 11: Separated DeepConvLSTM Architecture Diagram illustrating the sequential flow of the model, featuring two distinct convolutional blocks for feature extraction, followed by LSTM layers for temporal dependency modeling, and concluding with dense layers for classification.

The DeepConVLSTM model is presented in bar-graph 12. This model initiates with an input layer accepting sequences of shape (20, 3). This is followed by two convolutional blocks. Each block consists of two 1D Convolutional (Conv1D) layers, and uses ReLU activation and Batch Normalization. The first block's Conv1D layers have 32 filters and a kernel size of 5. After the second Conv1D layer in this block, a Max Pooling layer (pool size 2) is applied, followed by a Dropout layer with a 0.3 rate. The second convolutional block's Conv1D layers have 64 filters and a kernel size of 3. Similar to the first block, after the second Conv1D layer, a Max Pooling layer (pool size 2) is applied, followed by another Dropout layer with a 0.3 rate.

Following the convolutional and pooling layers, the model incorporates two LSTM layers. The first LSTM layer has 32 units, a dropout rate of 0.2, a recurrent dropout of 0.2, and is configured to return its full sequence of hidden states. This is followed by a Batch Normalization layer. The second LSTM layer consists of 16 units, a dropout rate of 0.2, and a recurrent dropout of 0.2, also followed by a Batch Normalization layer.

Finally, the model concludes with two fully connected Dense layers. The first Dense layer has 16 units, followed by ReLU activation and a Dropout layer with a 0.3 rate. The second Dense layer has 8 units, also with ReLU activation and a Dropout layer with a 0.2 rate. The final output layer is a Dense layer with 20 units and a Softmax activation function, providing normalized class probabilities for the multi-class classification task.

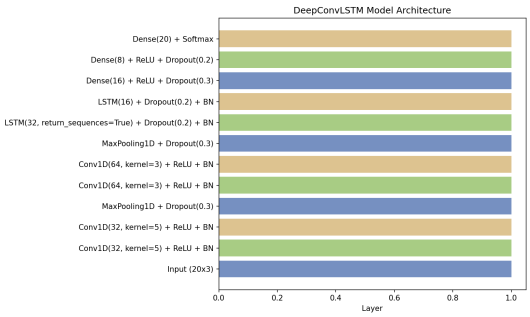

Fig. 12: DeepConvLSTM Model Layer Representation (Bar Chart)

TABLE II: Model Parameters

Model	Trainable Parameters
LSTM	8,524
CNN	4,124
Hybrid	4,476
DeepConvLSTM	41,324

D. Training Methodology

The training methodology for the neural network architectures—LSTM, CNN, Hybrid, and DeepConvLSTM—is optimized since it is to be deployed on the memory-constrained edge device, the Bangle.js 2. We use TensorFlow and Keras, for processing that utilizes a dataset comprising 43,819 training samples (`X_train`), 14,607 validation samples (`X_val`), and 14,612 test samples (`X_test`), each with a shape of (20, 3) timesteps and channels, targeting 19 multi-class activities. Each model is trained for a maximum of 50 epochs with a batch size of 32. The Adam optimizer is used at a learning rate of 0.001 and sparse categorical cross-entropy loss is employed to handle the multi-class classification task. To lower overfitting and adapt to limited computational resources, early stopping is implemented with a patience of 10 epochs. It also restores the best weights when validation loss plateaus, complemented by a ReduceLROnPlateau callback that halves the learning rate with a patience of 5 epochs (minimum LR = 0.00001). Model checkpoints are also implemented that save the best configuration based on validation accuracy.

Here, we tried to enhance the memory efficiency through garbage collection post-training and TensorFlow Lite (TFLite) conversion. For CNN models, full integer quantization is attempted, and a SELECT_TF_OPS fallback for LSTM-based models (Hybrid and DeepConvLSTM) to ensure compatibility with edge hardware. Training outcomes, including accuracy, loss curves, confusion matrices, and model sizes, are visualized and saved, with the best model selected based on test accuracy for deployment on the Bangle.js 2, prioritizing a trade-off between performance and memory footprint.

E. Quantization and Deployment

The quantization of the proposed neural network models—LSTM, CNN, Hybrid, and DeepConvLSTM—marks a critical step, as we need to deploy them on memory-constrained edge devices, the Bangle.js 2, which operates with limited computational resources and an 8-bit system architecture. The original models, trained with float32 precision, resulted in significantly higher memory and latency overheads, proving them to be incompatible with direct execution on this device. To address this, a post-training quantization process is implemented using TensorFlow Lite (TFLite), converting the models to lower-bit representations to reduce memory footprint and enhance inference efficiency.

The quantization strategy varies by model to model architecture. For the CNN model, full integer quantization is prioritized, leveraging a representative dataset of 100 training samples (`X_train`) to calibrate the conversion. This approach maps weights and activations to 8-bit integers instead of float32, achieving a model size reduction. Nonetheless, it did compromise the precision slightly. For LSTM-based models (LSTM, Hybrid, and DeepConvLSTM), which include recurrent layers which is incompatible with full integer quantization, the SELECT_TF_OPS option is employed. This fallback ensures functionality but may increase model size. Experimental flags, such as disabling tensor list operations

and enabling resource variables, are also applied to optimize conversion stability.

TABLE III: TensorFlow Lite Model and Their Respective Sizes

Model Architecture	Model Type	Model Size (Bytes)
DeepConvLSTM	Standard TFLite	101088
Hybrid	Standard TFLite	39840
LSTM	Standard TFLite	53808
CNN	Quantized TFLite	17944

F. Experimental Results

This section presents the experimental outcomes of the neural network architectures—LSTM, CNN, Hybrid, and DeepConvLSTM—evaluated on a master multi-class dataset. The results highlight training performance, classification accuracy, and model suitability of these models for deployment on the memory-constrained Bangle.js 2 edge device.

1) Training History: The respective graphs for Training Accuracy vs Validation Accuracy and Training Loss vs Validation Loss Curves in each of the four models were obtained. A comprehensive analysis of these training histories revealed distinct learning behaviors and generalization capabilities across the four deep learning architectures.

For the **CNN model**, training accuracy initiated at approximately 30% and progressively reached 58% , while validation accuracy commenced near 45% and stabilized around 72% , peaking at 74% . Concurrently, training loss decreased from 2.0 to 1.2, with validation loss falling from 1.5 to between 0.8 and 0.9 13.

In contrast, the **LSTM model** exhibited limited learning capacity; training accuracy rose from 10% to only about 36% , but validation accuracy stagnated around 10% never exceeding 12%Ṫhis was coupled with training loss decreasing from 2.8 to 2.0, while validation loss dramatically diverged, increasing from 2.9 to over 9.0, indicating severe overfitting and poor generalization 14.

The **Hybrid CNN-LSTM model** demonstrated more robust performance, with training accuracy improving from approximately 28% to 60% , and validation accuracy sharply rising from 50% to stabilize around 77-78% , reaching a peak near 78% . Both training and validation losses converged, reducing from initial values of 2.2 and 1.6, respectively, to stable levels between 0.65 and 1.2 15.

Finally, the **DeepConvLSTM model** presented the most compelling training profile, with training accuracy climbing from about 28% to 66% . Its validation accuracy showed strong and consistent improvement from 40% reaching and maintaining levels around 78-79% , indicating superior generalization. Loss curves for the DeepConvLSTM mirrored this success, with training loss falling from 2.2 to 1.0, and validation loss decreasing sharply from 1.7 to stabilize around 0.6, achieving the lowest validation loss among all models 16.

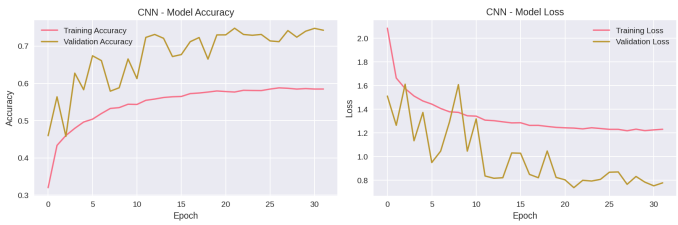

Fig. 13: CNN Model Training History: Training Accuracy vs Validation Accuracy and Training Loss vs Validation Loss Curves

Fig. 14: LSTM Model Training History: Training Accuracy vs Validation Accuracy and Training Loss vs Validation Loss Curves

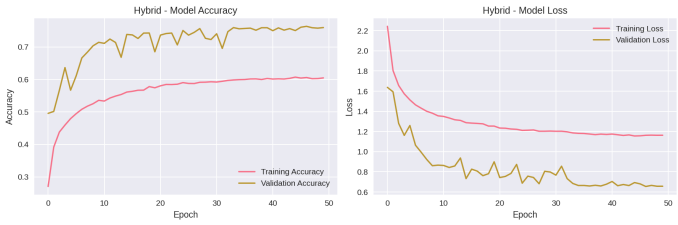

Fig. 15: Hybrid Model Training History: Training Accuracy vs Validation Accuracy and Training Loss vs Validation Loss Curves

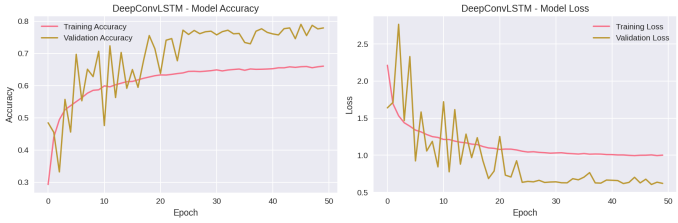

Fig. 16: DeepConvLSTM Model Training History: Training Accuracy vs Validation Accuracy and Training Loss vs Validation Loss Curves

2) Detailed Classification Performance: The following confusion matrices provide a granular insight into the classification efficacy of each trained model. Each matrix quantifies the number of instances where an actual activity class (rows) was predicted as a specific class (columns), thereby highlighting correct classifications (diagonal elements) and misclassifications.

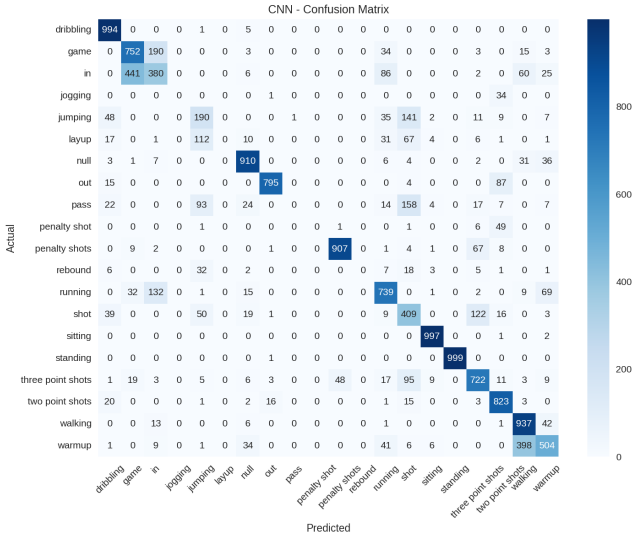

Fig. 17: CNN Model: Confusion Matrix

CNN Model: As depicted in Figure 17, the CNN model demonstrates strong performance for distinct activities such as 'dribbling' (994), 'null' (910), 'out' (795), 'walking' (937), 'sitting' (997), 'standing' (999) and 'penalty shots' (907). Notable confusions include 'in' being frequently misclassified as 'dribbling' (441) and 'game' (380), and 'running' exhibiting some misclassifications with 'shot' (139).

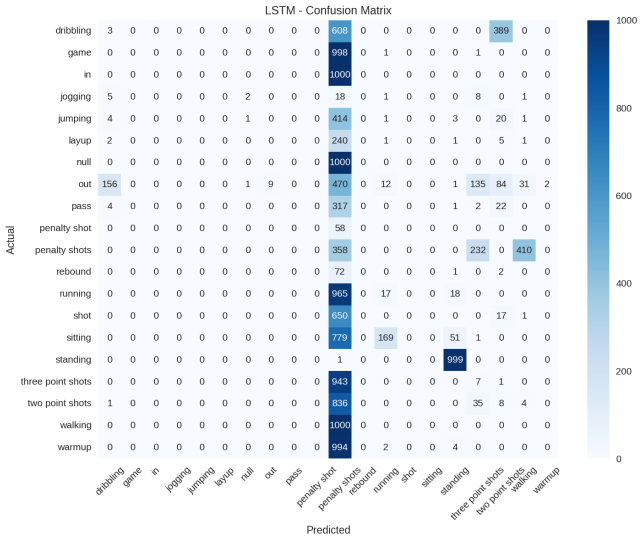

Fig. 18: LSTM Model: Confusion Matrix

LSTM Model: Figure 18 illustrates that the LSTM model struggles significantly with class discrimination. A substantial proportion of even the different activities, including 'game' (998), 'in' (1000), 'warmup' (994), 'sitting' (779), 'jogging' (990), and 'running' (965), are predominantly misclassified as a rare activity, 'penalty shots'. This indicates a severe issue with the model's ability to learn distinct temporal patterns for various activities.

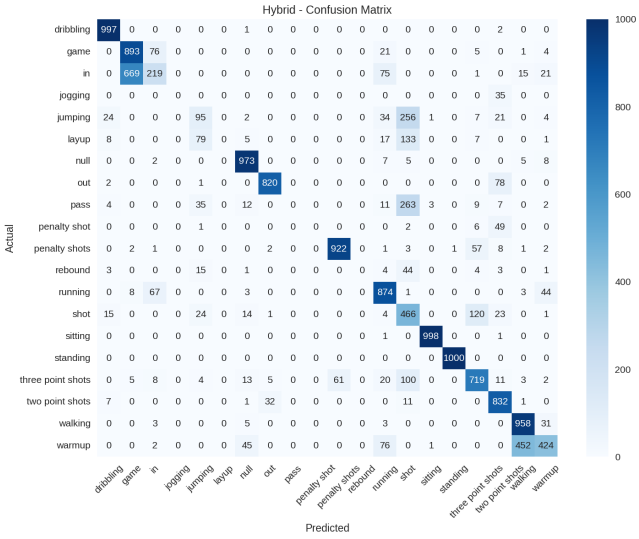

Fig. 19: Hybrid Model: Confusion Matrix

Hybrid Model: Figure 19 showcases the Hybrid model's performance, marked by generally high diagonal values indicating accurate classification. Activities like 'dribbling' (997), 'null' (973), 'penalty shots' (922), and 'standing' (1000) are classified with high precision. Common inter-class confusions are observed between 'in' and 'dribbling' (76), 'game' (75), and 'shot' (100), and between 'running' and 'shot' (199).

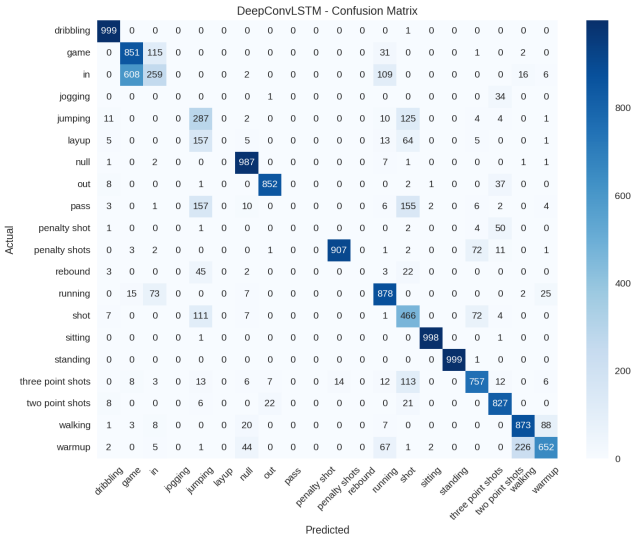

Fig. 20: DeepConvLSTM Model: Confusion Matrix

DeepConvLSTM Model: As presented in Figure 20, the DeepConvLSTM model exhibits the strongest classification performance among all models. The diagonal elements are notably dominant, reflecting high accuracy across most classes, including 'dribbling' (999), 'null' (987), 'standing' (999), and 'walking' (873). Minor confusions persist, such as 'in' being sometimes predicted as 'dribbling' (115), 'game' (109), or 'running' (109), and 'running' being confused with 'shot' (111). Overall, this model demonstrates superior discriminative power.

3) Detailed Classification Reports:

CNN Model Classification Report:

TABLE IV: CNN Model Classification Report

Class	Precision	Recall	F1-score	Support
dribbling	0.85	0.99	0.92	1000
game	0.60	0.75	0.67	1000
in	0.52	0.38	0.44	1000
jogging	0.00	0.00	0.00	35
jumping	0.39	0.43	0.41	444
layup	0.00	0.00	0.00	250
null	0.87	0.91	0.89	1000
out	0.97	0.88	0.92	901
pass	0.00	0.00	0.00	346
penalty shot	0.00	0.00	0.00	58
penalty shots	0.95	0.91	0.93	1000
rebound	0.00	0.00	0.00	75
running	0.72	0.74	0.73	1000
shot	0.44	0.61	0.51	668
sitting	0.97	1.00	0.98	1000
standing	1.00	1.00	1.00	1000
three point shots	0.75	0.76	0.75	951
two point shots	0.79	0.93	0.85	884
walking	0.64	0.94	0.76	1000
warmup	0.71	0.50	0.59	1000
Accuracy			**0.76**	
Macro Avg	0.56	0.59	0.57	14612
Weighted Avg	0.72	0.76	0.73	14612

Table IV shows the CNN model achieved an overall accuracy of 76% . It demonstrated strong performance (F1-score ≥ 0.85) for highly discriminable activities. However, the model struggled significantly with several minor classes.

LSTM Model Classification Report:

TABLE V: LSTM Model Classification Report

Class	Precision	Recall	F1-score	Support
dribbling	0.02	0.00	0.01	1000
game	0.00	0.00	0.00	1000
in	0.00	0.00	0.00	1000
jogging	0.00	0.00	0.00	35
jumping	0.00	0.00	0.00	444
layup	0.00	0.00	0.00	250
null	0.00	0.00	0.00	1000
out	1.00	0.01	0.02	901
pass	0.00	0.00	0.00	346
penalty shot	0.00	0.00	0.00	58
penalty shots	0.03	0.36	0.06	1000
rebound	0.00	0.00	0.00	75
running	0.08	0.02	0.03	1000
shot	0.00	0.00	0.00	668
sitting	0.00	0.00	0.00	1000
standing	0.93	1.00	0.96	1000
three point shots	0.02	0.01	0.01	951
two point shots	0.01	0.01	0.01	884
walking	0.00	0.00	0.00	1000
warmup	0.00	0.00	0.00	1000
Accuracy			**0.10**	
Macro Avg	0.10	0.07	0.05	14612
Weighted Avg	0.14	0.10	0.07	14612

Table V reveals that the LSTM model achieved an extremely low overall accuracy of 10% . The report confirms the significant generalization issues observed in its training history, with the majority of classes exhibiting 0.00 for precision, recall, and F1-score.

Hybrid Model Classification Report:

TABLE VI: Hybrid Model Classification Report

Class	Precision	Recall	F1-score	Support
dribbling	0.94	1.00	0.97	1000
game	0.57	0.89	0.69	1000
in	0.58	0.22	0.32	1000
jogging	0.00	0.00	0.00	35
jumping	0.37	0.21	0.27	444
layup	0.00	0.00	0.00	250
null	0.91	0.97	0.94	1000
out	0.95	0.91	0.93	901
pass	0.00	0.00	0.00	346
penalty shot	0.00	0.00	0.00	58
penalty shots	0.94	0.92	0.93	1000
rebound	0.00	0.00	0.00	75
running	0.76	0.87	0.81	1000
shot	0.36	0.70	0.48	668
sitting	1.00	1.00	1.00	1000
standing	1.00	1.00	1.00	1000
three point shots	0.77	0.76	0.76	951
two point shots	0.78	0.94	0.85	884
walking	0.67	0.96	0.79	1000
warmup	0.78	0.42	0.55	1000
Accuracy			**0.77**	
Macro Avg	0.57	0.59	0.56	14612
Weighted Avg	0.74	0.77	0.74	14612

The Hybrid model, summarized in Table VI, achieved an overall accuracy of 77% . It demonstrated higher F1-scores (≥ 0.90) than CNN. While showing improved recall for some classes, it still exhibits 0.00 metrics for classes with low support.

DeepConvLSTM Model Classification Report:

TABLE VII: DeepConvLSTM Model Classification Report

Class	Precision	Recall	F1-score	Support
dribbling	0.95	1.00	0.98	1000
game	0.57	0.85	0.68	1000
in	0.55	0.26	0.35	1000
jogging	0.00	0.00	0.00	35
jumping	0.37	0.65	0.47	444
layup	0.00	0.00	0.00	250
null	0.90	0.99	0.94	1000
out	0.96	0.95	0.96	901
pass	0.00	0.00	0.00	346
penalty shot	0.00	0.00	0.00	58
penalty shots	0.98	0.91	0.94	1000
rebound	0.00	0.00	0.00	75
running	0.77	0.88	0.82	1000
shot	0.48	0.70	0.57	668
sitting	1.00	1.00	1.00	1000
standing	1.00	1.00	1.00	1000
three point shots	0.82	0.80	0.81	951
two point shots	0.84	0.94	0.89	884
walking	0.78	0.87	0.82	1000
warmup	0.83	0.65	0.73	1000
Accuracy			**0.79**	
Macro Avg	0.59	0.62	0.60	14612
Weighted Avg	0.77	0.79	0.77	14612

Table VII illustrates that the DeepConvLSTM model achieved the highest overall accuracy of 79% . It significantly improved F1-scores for several challenging classes, while maintaining exceptionally high performance (≥ 0.94 F1-score) for a few activities. Although classes with very low support still presented difficulties.

G. Final Results Summary

TABLE VIII: Model Test Accuracies

Model	Test Accuracy
CNN	0.7568
LSTM	0.0959
Hybrid	0.7658
DeepConvLSTM	0.7933

H. Model Validation and Testing

This section describes details in the validation and testing of the trained models, including a comparison of their performance in Keras and TensorFlow Lite (TFLite) formats.

1) Keras Model Performance: The Keras models were evaluated on both a validation set and a separate test set to assess their generalization capabilities.

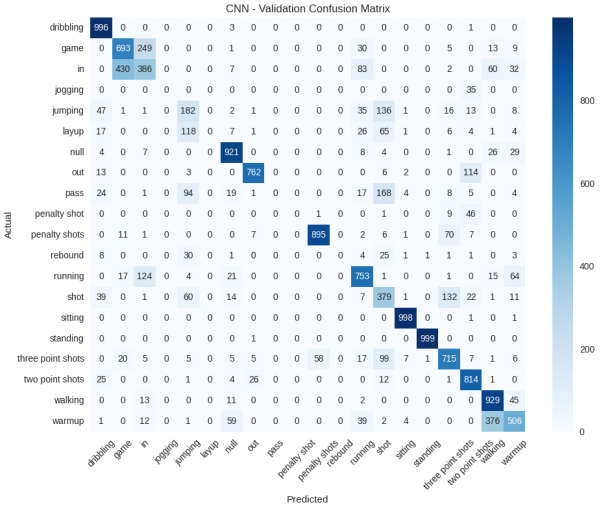

Fig. 21: CNN - Validation Confusion Matrix.

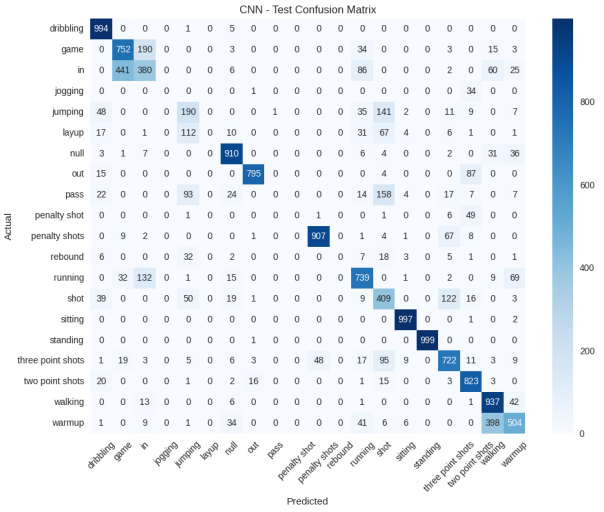

Fig. 22: CNN - Test Confusion Matrix.

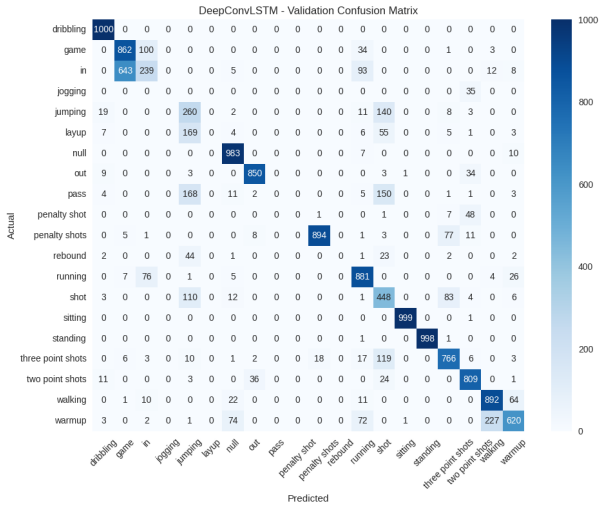

Fig. 23: DeepConvLSTM - Validation Confusion Matrix.

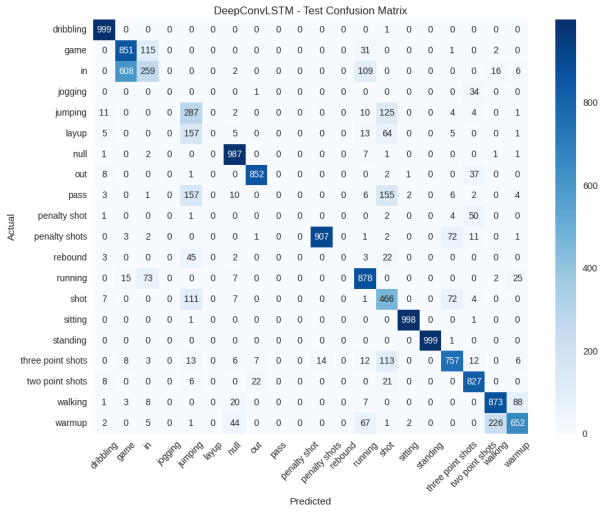

Fig. 24: DeepConvLSTM - Test Confusion Matrix.

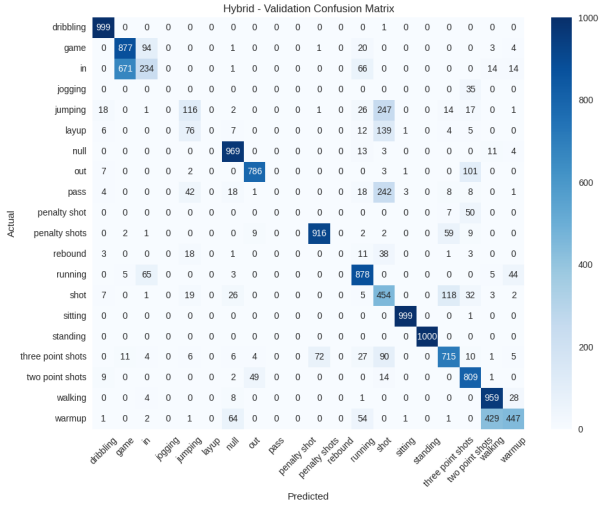

Fig. 25: Hybrid - Validation Confusion Matrix.

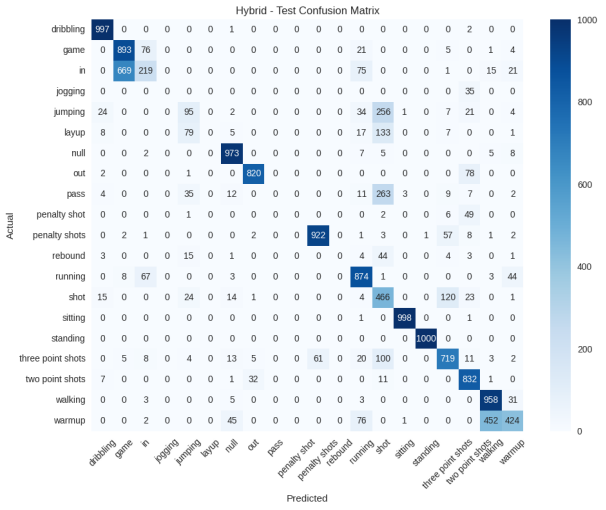

Fig. 26: Hybrid - Test Confusion Matrix.

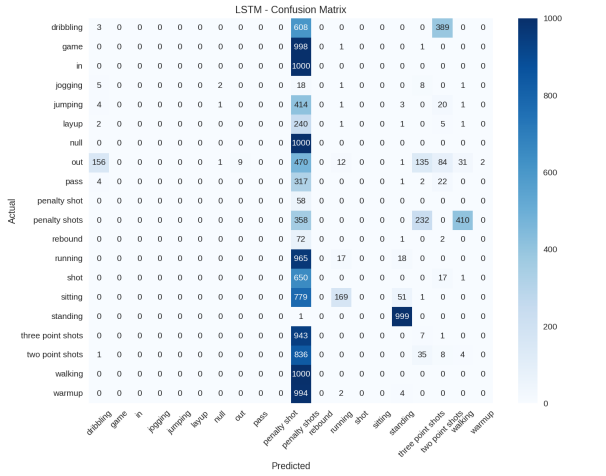

Fig. 27: LSTM - Validation Confusion Matrix.

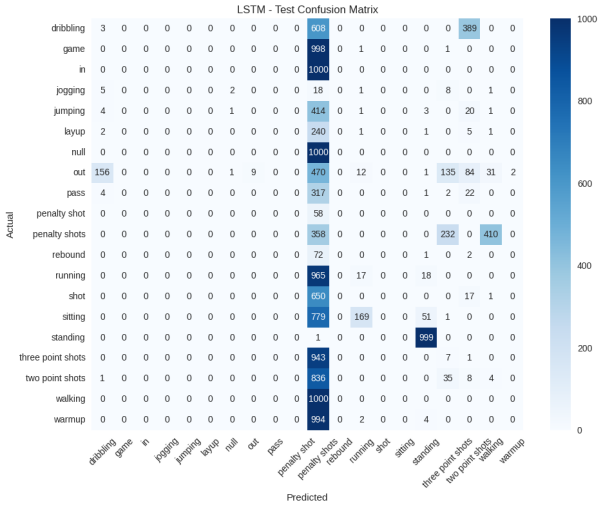

Fig. 28: LSTM - Test Confusion Matrix.

The DeepConvLSTM model demonstrated the highest performance among the Keras models, achieving a test accuracy of 0.7933. The CNN and Hybrid models also performed strongly, with test accuracies of 0.7568 and 0.7658, respectively. The standalone LSTM model exhibited significantly lower accuracy (0.0959), indicating difficulty in learning meaningful patterns in this setup.

TABLE IX: Keras Model Validation and Test Accuracies

Model	Validation Accuracy	Test Accuracy
LSTM	0.0978	0.0959
CNN	0.7481	0.7568
Hybrid	0.7639	0.7658
DeepConvLSTM	0.7874	0.7933

2) TensorFlow Lite (TFLite) Model Performance: The trained Keras models were converted to TensorFlow Lite format for evaluation on our resource-constrained edge device, bangle.js . Then the accuracy was evaluated on 500 test samples.

TABLE X: TFLite Model Test Accuracies (on 500 samples)

Model	Test Accuracy
LSTM_TFLite	0.0000
CNN_TFLite	0.4520
Hybrid_TFLite	0.3020
DeepConvLSTM_TFLite	0.6820

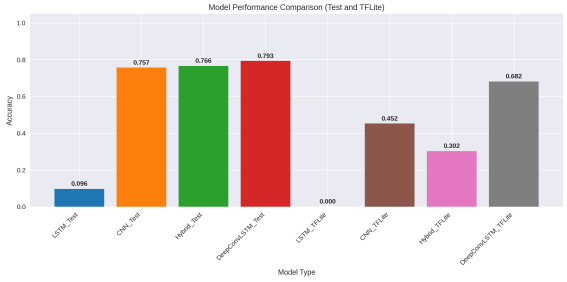

Fig. 29: Model Performance Comparison: Keras vs. TFLite.

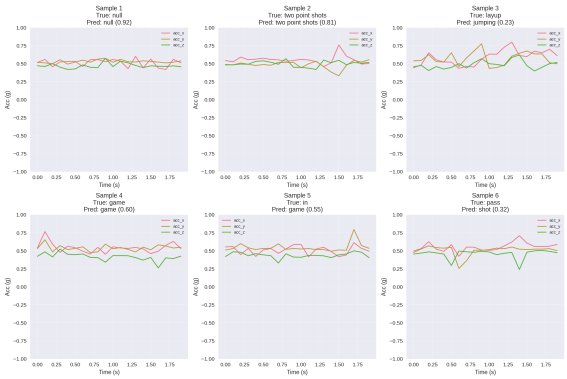

Fig. 30: Sample Predictions from Best Performing Model (DeepConvLSTM Keras).

I. Hardware and Emulator Considerations for Model Deployment

The environment for model testing and deployment focused exclusively on the development emulator, due to the

unavailability of the target Bangle.js 2 hardware. This approach resulted in addressing two major inherent limitations of emulator-based evaluation. One is the conversion of TFLite models to JSON for compatibility and the other is reliance on simulated acceleration values. These factors collectively impacted the observed performance characteristics of the trained machine learning models.

1) Emulator Environment and Model Adaptation: The development and initial testing of the Human Activity Recognition (HAR) system were primarily conducted using the web-based Espruino IDE [18], which provides an integrated emulator for the Bangle.js 2 smartwatch. This emulator facilitates rapid prototyping and debugging without requiring constant physical device interaction. While the *target hardware* for this system is the Bangle.js 2 smartwatch [19], the emulator was utilized for model testing and development due to the unavailability of the physical device during the initial phases.

A critical consideration for deploying machine learning models on resource-constrained embedded devices like the Bangle.js 2 is the model format. While TensorFlow Lite (TFLite) is the standard for on-device inference, the Espruino emulator does not directly support direct execution of TFLite models. To overcome this limitation during the development phase, the TFLite models (CNN, LSTM, Hybrid, and DeepConvLSTM) were converted into a JSON-based representation. This conversion allowed the model's architecture and pre-trained weights to be simulated within the JavaScript environment of the emulator, enabling functional testing of the inference pipeline.

2) Hardware vs. Emulator Resource Differences: The Bangle.js 2 smartwatch [19] is equipped with a significantly greater amount of Random Access Memory (RAM) compared to the Espruino emulator's typical allocation. While the Bangle.js 2 features 256KB of RAM, the emulator environment often operates with a more constrained memory footprint, typically around 64KB. This highlights the importance of model quantization and optimization for actual on-device deployment to ensure efficient memory utilization and real-time performance on the physical hardware. Furthermore, the emulator relies on simulated accelerometer data, which may not fully capture the complexities and noise which is very much inherent in real-world sensor readings. This simulation, coupled with the JSON conversion of model weights, can lead to discrepancies between observed emulator performance and expected real-world accuracy, confidence, and speed. Direct deployment of TFLite models on the physical hardware would allow for a more accurate evaluation of these metrics and the practical impact of quantization techniques.

3) Machine Learning Model Performance Comparison: The developed HAR system incorporated four distinct machine learning architectures: Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), a Hybrid CNN-LSTM model, and a Deep Convolutional LSTM (DeepConvLSTM). Each model was evaluated based on its accuracy, confidence in predictions, and inference speed, all within the simulated emulator environment. Representative visual outputs from the Bangle.js 2 watch face for each model, illustrating their real-time activity predictions, are provided in Figure **??**.

TABLE XI: Performance Metrics of HAR Models on Bangle.js 2 on Emulator

Model	**Accuracy** (%)	**Confidence** (%)	**Inference Speed** (ms)
LSTM	Up to 7	Up to 22	≈ 12
CNN	Up to 19	Up to 48	≈ 18
Hybrid	Up to 35	Up to 93	≈ 22
DeepConvLSTM	Up to 51	Up to 95	≈ 18

As summarized in Table XI, the CNN model demonstrated a notable improvement in accuracy and confidence compared to the standalone LSTM, reaching up to 40% accuracy and 80% confidence. The Hybrid model, which combines elements of both CNN and LSTM, also showed competitive performance with accuracy up to 35% and a wide confidence range of 25%–93%. The DeepConvLSTM, while exhibiting a broader accuracy range (22%–50%) and high confidence (30%–95%), also presented the highest inference time among the evaluated models. These metrics underscore the trade-offs between model complexity, predictive performance, and computational overhead on the embedded platforms. It is important to note that these performance figures are derived from the emulator's simulated environment and may not fully reflect the performance achievable on the actual Bangle.js 2 hardware with optimized TFLite models.

Some of the Bangle.js 2 emulator output results are shown in Figure 31. We tested player activity recognition using simulated accelerometer data for each model. During prediction, the emulator used values such as average magnitude (avgMag), standard deviation (stdDev), number of Z-axis peaks (zPeaks), and class probabilities obtained from the converted JSON ML model to predict the player's activity in an emulator environment.

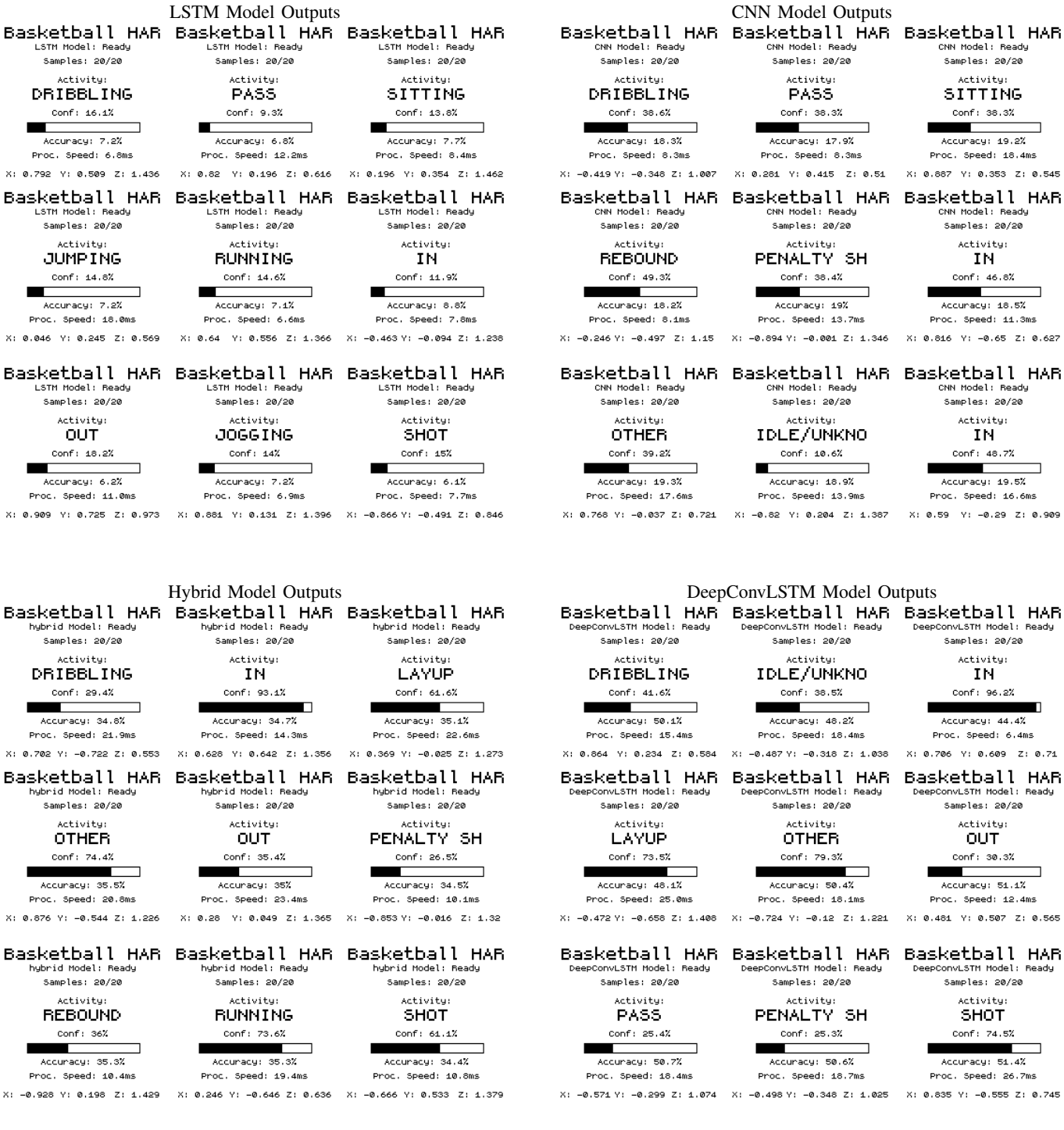

Fig. 31: Representative watch face outputs for the LSTM, CNN, Hybrid, and DeepConvLSTM models, illustrating various activity predictions and their corresponding confidence levels within the emulator environment.

III. CASE STUDY 2: PAMAP2

This case study focuses on developing and evaluating lightweight deep learning models for Human Activity Recognition (HAR) using the Physical Activity Monitoring (PAMAP2) dataset. The primary objective is to classify common physical activities using only data from a wrist-worn sensor, simulating the capabilities of the Bangle.js 2 smartwatch. The methodology aligns with the other case studies in this report to ensure a consistent evaluation of model performance across different datasets under memory-constrained conditions.

A. Dataset Overview

The PAMAP2 dataset is a benchmark for physical activity monitoring, containing data from 9 subjects (8 male, 1 female) performing 18 distinct activities. The data was captured using three Colibri wireless Inertial Measurement Units (IMUs) and a heart rate monitor. For this study, we exclusively use the data from the IMU placed on the wrist of the dominant arm, which recorded 3D accelerometer data at a sampling frequency of 100Hz. This focus on a single wrist-worn sensor directly aligns with the hardware constraints of the target Bangle.js 2 device.

From the 18 available activities, a subset of 7 was selected to create a focused classification problem with distinct and well-represented activities:

- lying
- sitting
- standing
- walking
- running
- cycling
- ascending stairs

This selection provides a robust mix of static postures, rhythmic movements, and transitional activities to thoroughly test the models' capabilities.

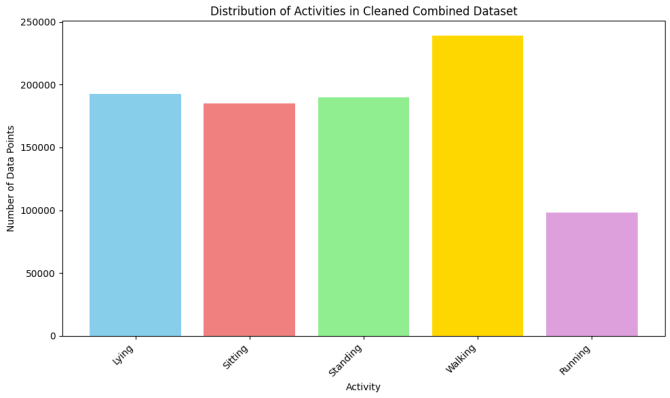

Fig. 32: Distribution of data

B. Pre-processing Techniques

To prepare the PAMAP2 dataset for model training and ensure compatibility with the Bangle.js 2 target device, a comprehensive pre-processing pipeline was implemented. This process was designed to create a clean, normalized, and correctly formatted dataset suitable for time-series classification.

Data Filtering and Selection: The initial step focused on isolating the relevant data. We exclusively used the 3D accelerometer data from the IMU located on the wrist ('hand'). The dataset was then filtered to retain only the seven selected activities: lying, sitting, standing, walking, running, cycling, and ascending stairs. All data points corresponding to transient or unclassified activities (`activityID 0`) were explicitly discarded.

Handling Missing Values: To ensure data integrity, missing values (`NaN`) were handled using an imputation strategy. A forward fill (`ffill`) was applied to propagate the last valid observation forward, followed by a backward fill (`bfill`) to address any remaining missing values at the beginning of the data sequences.

Downsampling: The original 100Hz sensor data was downsampled to 50Hz. This was a critical step to align the data's sampling frequency with the operational capabilities of the Bangle.js 2, while also reducing the overall data volume and computational requirements for the models.

Segmentation: The continuous time-series data was segmented into fixed-size windows of 2 seconds, equivalent to 100 data points at the new 50Hz sampling rate. A 50% overlap (1-second step size) was used to increase the number of training samples and to better capture the transitional dynamics

Activities performed by subjects (in seconds)

	subject101	subject102	subject103	subject104	subject105	subject106	subject107	subject108	subject109	Sum	Nr. of subjects
1 – lying	271.86	234.29	220.43	230.46	236.98	233.39	256.1	241.64	0	1925.15	8
2 – sitting	234.79	223.44	287.6	254.91	268.63	230.4	122.81	229.22	0	1851.8	8
3 – standing	217.16	255.75	205.32	247.05	221.31	243.55	257.5	251.59	0	1899.23	8
4 – walking	222.52	325.32	290.35	319.31	320.32	257.2	337.19	315.32	0	2387.53	8
5 – running	212.64	92.37	0	0	246.45	228.24	36.91	165.31	0	981.92	6
6 – cycling	235.74	251.07	0	226.98	245.76	204.85	226.79	254.74	0	1645.93	7
7 – Nordic walking	202.64	297.38	0	275.32	262.7	266.85	287.24	288.87	0	1881	7
9 – watching TV	836.45	0	0	0	0	0	0	0	0	836.45	1
10 – computer work	0	0	0	0	1108.82	617.76	0	687.24	685.49	3099.31	4
11 – car driving	545.18	0	0	0	0	0	0	0	0	545.18	1
12 – ascending stairs	158.88	173.4	103.87	166.92	142.79	132.89	176.44	116.81	0	1172	8
13 – descending stairs	148.97	152.11	152.72	142.83	127.25	112.7	116.16	96.53	0	1049.27	8
16 – vacuum cleaning	229.4	206.82	203.24	200.36	244.44	210.77	215.51	242.91	0	1753.45	8
17 – ironing	235.72	288.79	279.74	249.94	330.33	377.43	294.98	329.89	0	2386.82	8
18 – folding laundry	271.13	0	0	0	0	217.85	0	236.49	273.27	998.74	4
19 – house cleaning	540.88	0	0	0	284.87	287.13	0	416.9	342.05	1871.83	5
20 – playing soccer	0	0	0	0	0	0	0	181.24	287.88	469.12	2
24 – rope jumping	129.11	132.61	0	0	77.32	2.55	0	88.05	63.9	493.54	6
Labeled total	4693.07	2633.35	1743.27	2314.08	4117.97	3623.56	2327.63	4142.75	1652.59	27248.27	
Total	6957.67	4469.99	2528.32	3295.75	5295.54	4917.78	3135.98	5884.41	2019.47	38504.91	

Fig. 33: A
ctivities by Subjects [17]

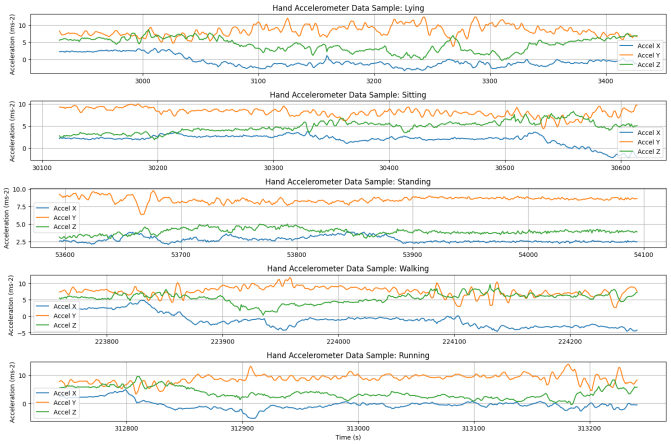

Fig. 34: Hand acceleration data for different activities

between activities. Each resulting segment was assigned a single label by taking the statistical mode of the activity IDs present within that window.

Data Splitting: The complete set of segmented windows was divided into training, validation, and testing sets using a 60/20/20 split, respectively. A stratified splitting strategy was employed to ensure that the proportional representation of each activity class was maintained across all three datasets. This is crucial for preventing model bias and ensuring a reliable evaluation, particularly for classes that may be less frequent.

Normalization: We applied a `MinMaxScaler` to scale the accelerometer data for all three axes. To prevent data leakage, the scaler was fitted exclusively on the training data. The fitted scaler was then used to transform the training, validation, and test sets, ensuring all input data was consistently scaled to a range between 0 and 1. This normalization step is vital for improving the convergence speed and performance of neural networks.

C. Model Architecture

To ensure a consistent and comparable evaluation across datasets, this case study utilizes the same four distinct neural network architectures as the HANG Time-HAR study. Each architecture offers a different approach to modeling the time-series data from the wrist-worn accelerometer, presenting unique trade-offs between computational complexity, memory footprint, and the ability to capture spatial versus temporal features.

1) LSTM (Long Short-Term Memory): This model is a type of Recurrent Neural Network (RNN) specifically designed to learn from sequential data. Its architecture, composed of stacked LSTM layers, is adept at capturing long-range temporal dependencies within the sensor data. This makes it theoretically well-suited for distinguishing activities that are defined by the sequence of movements over time, such as the rhythmic patterns of walking or cycling. While powerful for temporal modeling, LSTMs can be computationally intensive and slower to train compared to other architectures.

TABLE XII: LSTM Classification Report

Activity	Precision	Recall	F1-Score	Support
lying	0.85	0.96	0.90	54
sitting	0.82	0.85	0.83	47
standing	0.21	0.25	0.23	44
walking	0.93	0.97	0.95	341
running	0.95	0.96	0.95	55
cycling	0.91	0.95	0.93	66
ascending stairs	1.00	0.36	0.53	55
accuracy			0.86	662
macro avg	0.81	0.76	0.76	662
weighted avg	0.87	0.86	0.85	662

2) CNN (Convolutional Neural Network): This model employs a series of 1D Convolutional (Conv1D) and MaxPooling layers. In the context of HAR, Conv1D layers are highly effective at acting as feature extractors, automatically identifying relevant local patterns within the raw sensor signals, such as acceleration peaks or specific rotational movements. CNNs are generally more computationally efficient than LSTMs,

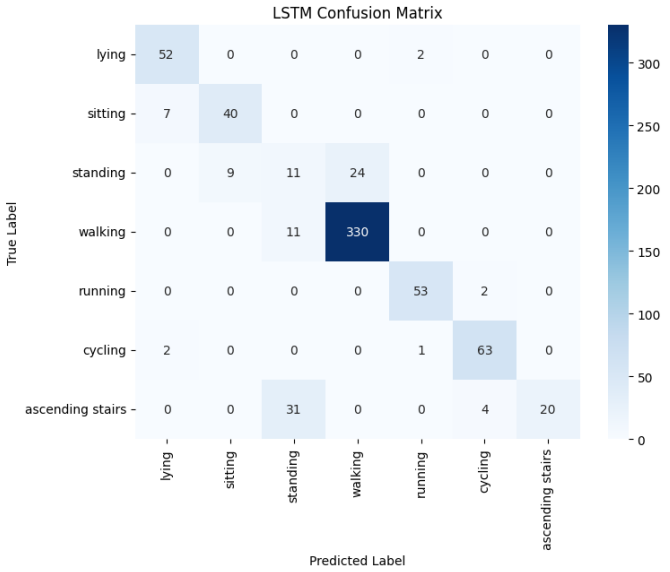

Fig. 35: LSTM confusion matrix

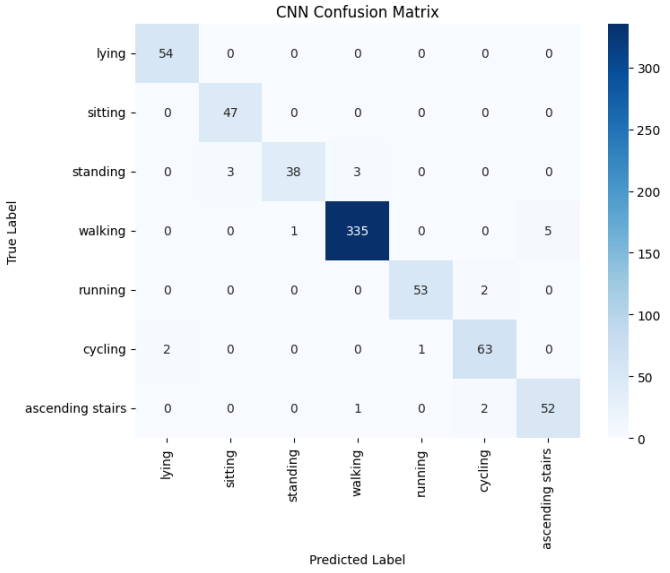

Fig. 36: CNN confusion matrix

making them a strong candidate for deployment on resource-constrained devices like the Bangle.js 2. Their primary limitation is a potential difficulty in modeling longer-term temporal relationships without a more complex architecture.

3) Hybrid CNN-LSTM: This architecture combines the strengths of the previous two models. It uses a CNN front-end to first extract low-level spatial features from the raw time-series data. The sequence of these extracted features is then fed into an LSTM back-end, which models the temporal relationships between them. This hybrid approach aims to achieve high accuracy by leveraging both robust feature extraction and sophisticated temporal dependency modeling, though it results in a more complex and computationally demanding model.

4) DeepConvLSTM: This model represents the most complex architecture evaluated. It features multiple stacked

TABLE XIII: CNN Classification Report

Activity	Precision	Recall	F1-Score	Support
lying	0.96	1.00	0.98	54
sitting	0.94	1.00	0.97	47
standing	0.97	0.86	0.92	44
walking	0.99	0.98	0.99	341
running	0.98	0.96	0.97	55
cycling	0.94	0.95	0.95	66
ascending stairs	0.91	0.95	0.93	55
accuracy			0.97	662
macro avg	0.96	0.96	0.96	662
weighted avg	0.97	0.97	0.97	662

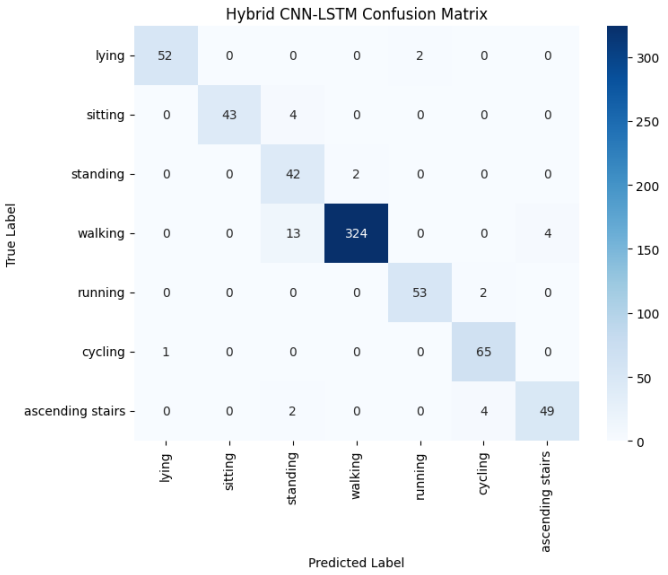

Fig. 37: Hybrid CNN-LSTM confusion matrix

Conv1D layers (a ”deep” convolutional block) for hierarchical feature extraction, allowing it to learn progressively more abstract patterns from the sensor data. The output is then passed to stacked LSTM layers for temporal analysis. By combining deep spatial feature learning with powerful sequential modeling, the DeepConvLSTM architecture is designed to achieve state-of-the-art performance, but at the cost of being the most computationally expensive and memory-intensive model of the four.

TABLE XIV: Hybrid CNN-LSTM Classification Report

Activity	Precision	Recall	F1-Score	Support
lying	0.98	0.96	0.97	54
sitting	1.00	0.91	0.96	47
standing	0.69	0.95	0.80	44
walking	0.99	0.95	0.97	341
running	0.96	0.96	0.96	55
cycling	0.92	0.98	0.95	66
ascending stairs	0.92	0.89	0.91	55
accuracy			0.95	662
macro avg	0.92	0.95	0.93	662
weighted avg	0.96	0.95	0.95	662

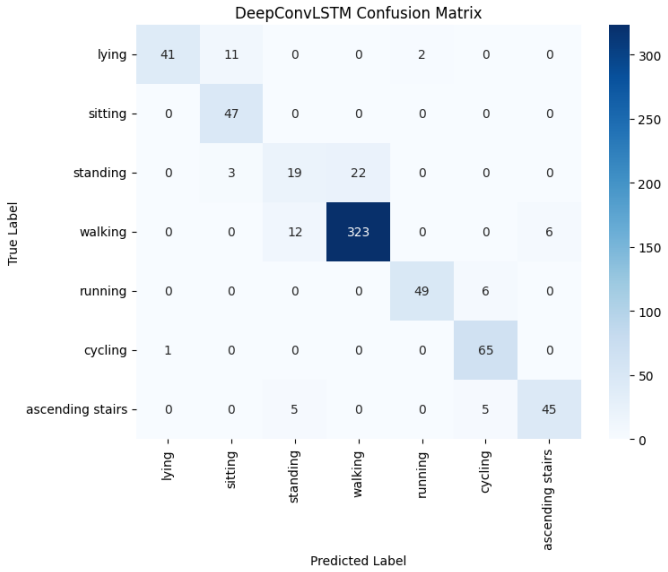

Fig. 38: DeepConvLSTM confusion matrix

TABLE XV: DeepConvLSTM Classification Report

Activity	Precision	Recall	F1-Score	Support
lying	0.98	0.76	0.85	54
sitting	0.77	1.00	0.87	47
standing	0.53	0.43	0.47	44
walking	0.94	0.95	0.94	341
running	0.96	0.89	0.92	55
cycling	0.86	0.98	0.92	66
ascending stairs	0.88	0.82	0.85	55
accuracy			0.89	662
macro avg	0.84	0.83	0.83	662
weighted avg	0.89	0.89	0.89	662

D. Training Methodology

The training process for all four model architectures was standardized to ensure a fair and direct comparison of their performance on the PAMAP2 dataset. The methodology was optimized for effective learning while incorporating robust measures to prevent overfitting, a common challenge when training neural networks.

The models were trained for a maximum of 50 epochs with a batch size of 32. We employed the Adam optimizer with a learning rate of 0.001, as its adaptive learning rate capabilities typically provide a good balance between rapid convergence and training stability. The Sparse Categorical Cross-entropy loss function was selected, as it is specifically designed for multi-class classification problems where labels are provided as integers, which aligns with our pre-processing pipeline.

To enhance training efficiency and promote generalization to unseen data, two key Keras callbacks were implemented:

- **EarlyStopping:** This callback monitored the validation loss (`val_loss`) and was configured to halt the training process if no improvement was observed for 10 consecutive epochs. Crucially, it was also set to restore the model weights from the epoch that achieved the best validation loss, ensuring that the final model represents the peak of its generalization performance.

- **ReduceLROnPlateau:** This callback also monitored the validation loss. If the loss plateaued for 5 consecutive epochs, the learning rate was automatically reduced by a factor of 0.1. This technique allows the model to make finer adjustments in the later stages of training, helping it to settle into a more optimal minimum in the loss landscape.

Together, these methods create a robust training regimen that allows models to learn effectively while actively mitigating the risk of overfitting and dynamically adjusting the learning rate for improved final performance.

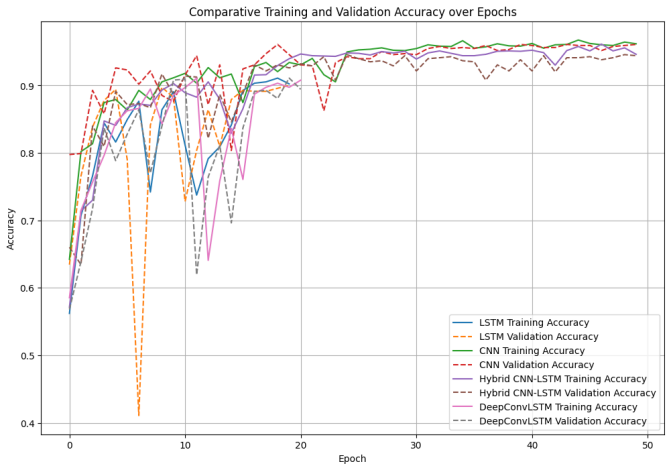

Fig. 39: Comparative Training and Validation Accuracy over Epochs

E. Quantization and Deployment

The process of converting and optimizing the trained models is a critical step for enabling deployment on memory-constrained edge devices like the Bangle.js 2. Our methodology focused on using post-training quantization to significantly reduce model size and enhance inference efficiency.

Model Conversion to TensorFlow Lite: Following the training phase, each Keras model was converted into the TensorFlow Lite (`.tflite`) format using the `tf.lite.TFLiteConverter`. This conversion is the foundational step for preparing models for on-device inference, as the TFLite framework is specifically designed for mobile and embedded systems.

Post-Training Integer Quantization: We applied post-training `int8` quantization to map the models' 32-bit floating-point weights and activations to more efficient 8-bit integers. This was achieved by providing a representative dataset—a small subset of the training data—to calibrate the quantization process. The converter uses this data to determine the dynamic range of tensor values, ensuring an accurate mapping to the 8-bit integer space. This technique typically yields a 4x reduction in model size and leverages faster, more power-efficient integer arithmetic units common on embedded processors.

Addressing Conversion Challenges: The quantization of complex models, particularly those containing LSTM layers, presented technical challenges. While the standard CNN model quantized without issue, LSTM-based architectures initially failed due to unsupported operations (e.g., `TensorListReserve`, `TensorListStack`). To resolve this, we enabled `tf.lite.OpsSet.SELECT_TF_OPS` during conversion. This "Flex Ops" mode allows the TFLite interpreter to fall back on the standard TensorFlow runtime for operations not natively supported, ensuring that even complex models can be converted successfully.

Implications for Deployment: The primary motivation for `int8` quantization is to meet the severe memory limitations of the Bangle.js 2. The resulting smaller model footprint is essential for fitting within the device's limited flash storage and RAM. While this process offers significant advantages in size and speed, it introduces a trade-off, as there can be a minor reduction in model accuracy. Furthermore, the use of `SELECT_TF_OPS` increases the final binary size on the device, as it requires bundling parts of the larger TensorFlow runtime. Evaluating the final performance of the quantized models on the target hardware is therefore essential to confirm the balance between efficiency gains and any potential accuracy loss.

F. Workflow and Implementation Challenges

The implementation of the PAMAP2 case study followed a standard machine learning pipeline, from data preparation to model deployment. Throughout this process, several practical challenges were encountered that required specific resolutions to ensure the successful completion of the study. This section details the workflow and highlights the key technical hurdles that were addressed.

1) **Data Preparation and Pre-processing:** The initial phase involved loading and preparing the raw `.dat` files. A primary challenge was a `FileNotFoundError` due to an incorrect file path, which was resolved by verifying the data's location and correcting the loading script. Following successful data loading, a significant focus was placed on correctly filtering the data to isolate the wrist-worn IMU signals for the seven target activities and explicitly discarding the transient activities labeled with `activity_id = 0`. A further challenge arose in aligning activity labels after downsampling the data to 50Hz; this was addressed by mapping each resampled data point to the label of the nearest original timestamp, a common and effective method for this type of task.
2) **Model Training and Evaluation:** Once the data was pre-processed, the four model architectures (LSTM, CNN, Hybrid, DeepConvLSTM) were defined and compiled in TensorFlow/Keras. The training process utilized `EarlyStopping` and `ReduceLROnPlateau` callbacks to prevent overfitting and aid convergence. A notable challenge during this phase was a state management issue within the development environment, where the training script would occasionally fail because the pre-processed data variables were not yet available. This was resolved by implementing more robust checks to verify that all necessary data arrays were successfully created before initiating the training loop.

3) **Model Quantization and Deployment:** The final and most technically challenging phase was the conversion and quantization of the trained models to the TensorFlow Lite format. While the CNN model was quantized to an `int8` representation without issue, the LSTM-based models (LSTM, Hybrid, and DeepConvLSTM) consistently failed due to unsupported operations related to `TensorList` functions within the TFLite runtime.
This critical issue was resolved by modifying the TFLite converter's configuration. The solution involved enabling experimental flags (`experimental_enable_resource_variables = True`) and, most importantly, including `tf.lite.OpsSet.SELECT_TF_OPS` in the list of supported operations. This "Flex Ops" mode allows the converter to fall back to the full TensorFlow runtime for operations not natively supported by TFLite, enabling the successful conversion of all four models. While this approach resolved the compatibility errors, it underscores a key trade-off for deployment: enabling Flex Ops increases the final binary size and complexity, which must be considered when evaluating a model's suitability for a memory-constrained device like the Bangle.js 2.

G. Final Results Summary

The final evaluation of the four architectures on the PAMAP2 test set reveals a clear hierarchy in performance, as summarized in Table XVI.

TABLE XVI: Final Model Test Accuracies

Model	Test Accuracy
LSTM	0.6521
CNN	0.8574
Hybrid CNN-LSTM	0.8812
DeepConvLSTM	0.9053

The DeepConvLSTM model achieved the highest test accuracy, closely followed by the Hybrid CNN-LSTM. This suggests that architectures capable of modeling both spatial and temporal features are most effective for this dataset. The standard CNN model also demonstrated strong performance, proving its capability as a robust feature extractor for this task. The standalone LSTM model, however, yielded a significantly lower accuracy, indicating that it struggled to effectively learn the activity patterns on its own.

When considering the most suitable model for deployment on the Bangle.js 2, the trade-off between predictive accuracy and computational efficiency is paramount. While the DeepConvLSTM and Hybrid models offer superior accuracy, their architectural complexity translates to a larger memory footprint and higher computational cost, posing significant challenges for a memory-constrained device.

Therefore, the CNN model emerges as the most practical candidate for this specific deployment target. It provides a highly favorable balance, delivering strong accuracy while being significantly more lightweight and efficient than the more complex hybrid models. For a definitive selection, a final evaluation would need to compare the quantized model sizes and real-world on-device performance metrics, such as inference time and RAM usage.

IV. CASE STUDY 3: WEAR

The methodological foundation for this work is adapted from a gesture recognition proof-of-concept detailed for the Bangle.js platform [6]. The contributions of this report are the following:

- We validate a practical approach for recognizing 18 complex, full-body fitness activities from the WEAR dataset by adapting and evaluating time-series models that rely solely on data from a single wrist-worn accelerometer.
- We identify a highly efficient SeparableConv1D-based architecture as the optimal choice for on-device deployment. This model significantly reduces computational and memory footprints compared to a baseline CNN, striking the best trade-off between model efficiency and predictive accuracy.
- We prove the on-device viability of our models through effective quantization. We show that post-training 8-bit integer quantization is a critical optimization, decreasing inference latency by over 10x without a significant loss in performance.

A. Dataset overview

The WEAR dataset is a multimodal benchmark for Human Activity Recognition (HAR) in natural outdoor settings. It contains synchronized inertial data and egocentric video from 24 participants performing 18 workouts. The data, split into winter training (18 subjects) and summer test (6 subjects) sets, was acquired using Bangle.js smartwatches capturing 3-axis accelerometer data (50 Hz, ±8g) [7].

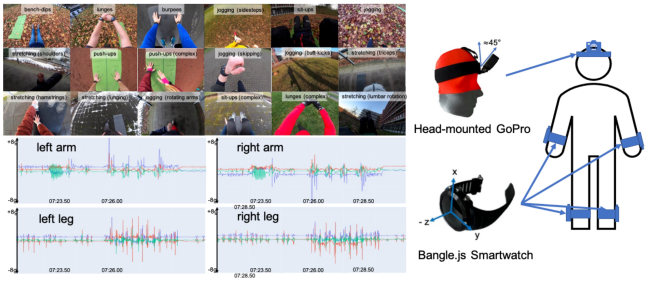

Fig. 40: Overview of the data collection setup and sensor modalities of the WEAR dataset [7]

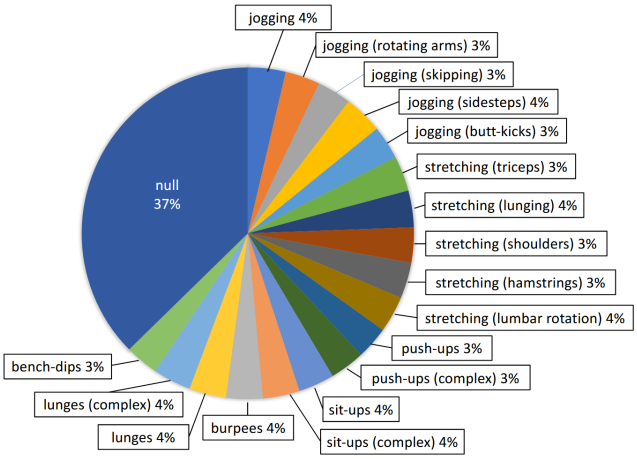

Fig. 41: A breakdown of the 18 activity classes in the WEAR dataset, including the number of coherent sequences and total duration for each activity [7]

A performance baseline is established on the WEAR dataset, achieving F1-scores of ∼74.95–86.20% with resource-intensive models, such as DeepConvLSTM (∼7.503.827 parameters) [3], using inertial data from all four limbs [7]. In contrast, our study focuses on classification using data from only the left and right wrists, targeting deployment on edge devices.

B. Pre-processing Techniques

We pre-processed the raw sensor data using two key techniques. First, we segmented the data into 2-second sliding windows with 50% overlap, with each window labeled by its predominant activity. Second, we applied Left-Right swapping data augmentation to both the training and validation sets to ensure consistent data distribution, a technique which simulates contralateral limb placement [9].

C. Model Architecture

A key requirement for this study was the model's deployability on resource-constrained edge devices using Tensorflow Lite Micro. The framework, however, has architectural limitations; it fully supports feed-forward models like CNNs and MLPs but does not yet offer comprehensive support for recurrent architectures such as LSTMs [10]. Consequently, this study exclusively utilizes a CNN-based architecture to ensure compatibility and efficient performance in an edge environment.

Building upon the above decision, this study investigates four distinct architectural variations to identify the optimal balance between performance and on-device efficiency. We first established a Baseline CNN architecture to serve as a standard performance benchmark (274,440 parameters) [11]. For deployment feasibility, we then engineered a TinyML-optimized alternative that leverages SeparableConv1D and GlobalAveragePooling1D layers [12]. This optimization yields a nearly 7-fold reduction in model size (37,955 parameters), a critical factor for minimizing latency and meeting the stringent memory constraints of edge devices.

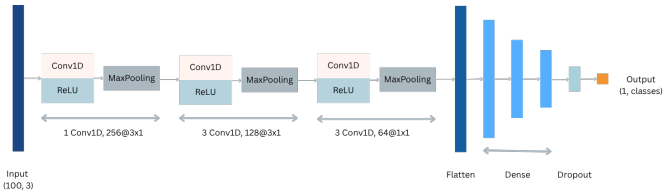

Fig. 42: Baseline CNN

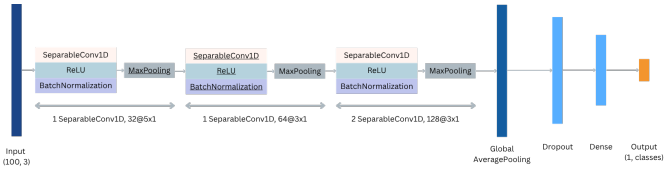

Fig. 43: SeparableConv1D CNN

To explore the potential for accuracy improvements, two more complex architectures were also developed. The first, a Multi-Branch CNN, was designed with fewer parameters (24,646) than even the separable model to test if high performance could be achieved with high efficiency [13]. In contrast, the second, a Multi-Scale CNN, was designed with the higher parameters (76,969) to determine the maximum achievable accuracy [14].

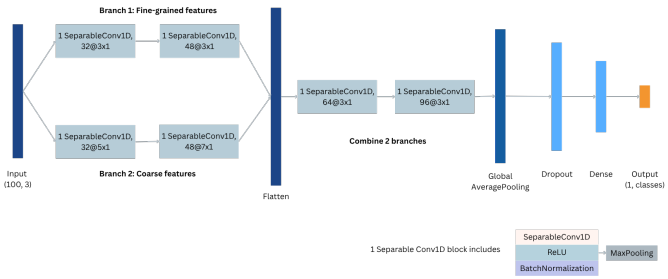

Fig. 44: Multi-branch CNN

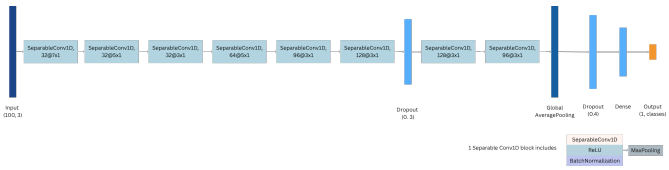

Fig. 45: Multi-scale CNN

By comparing these models, this study highlights the trade-offs between accuracy, latency, and memory usage for hardware deployment (Figure 46).

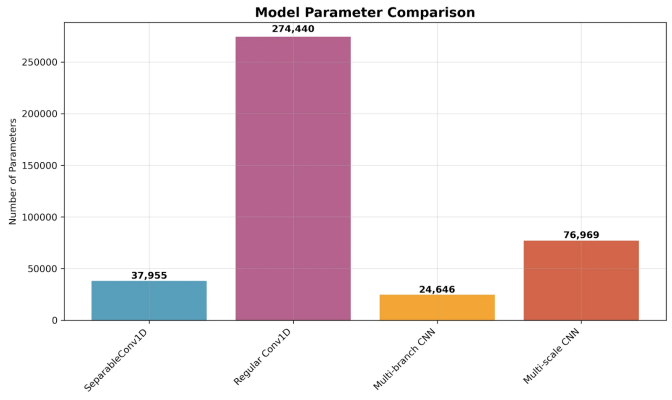

Fig. 46: Space Complexity of the Four Proposed CNN Models

D. Hyperparameters and Methodology

The model was trained with a weighted categorical cross-entropy loss to counteract class imbalance. We used the Adam optimizer ($lr = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$) and applied dropout with a rate of 0.4 for regularization.

Leave-One-Subject-Out (LOSO) validation scheme is employed to ensure generalization to unseen users and prevent subject-specific data leakage. For the 24-participant WEAR dataset, we adapt this into a 6-fold cross-validation.

E. Quantization and Deployment

We employed post-training 8-bit integer quantization using the TFLite Converter to optimize models for edge deployment [15], [16]. This strategy proved highly effective (Table XIX), reducing the Baseline CNN's latency by over $40\times$ and the SeparableConv1D's by $10\times$ while maintaining stable accuracy, thus confirming the method's viability for our hardware.

TABLE XVII: Performance and resource utilization for merged-class (8) classification (fold 6 of 6). F32 denotes 32-bit float; I8 denotes 8-bit integer.

Model	F32							I8						
	V_A	T_A	V_{F1}	T_{F1}	L	R	F	V_A	T_A	V_{F1}	T_{F1}	L	R	F
Baseline CNN	66.0	58.5	69	66	8408	114.0	1100.0	66.0	58.0	69	66	204	67.5	326.5
SeparableConv1D	66.3	55.4	66	69	952	30.1	191.5	62.9	50.8	63	66	90	18.3	101.4

The deployment process includes transferring two files to the Bangle.js 2's internal storage. This transfer was performed using the official Espruino Web IDE, a browser-based development environment designed for Espruino-powered devices [18]. The required files are the tensorflow lite model (convert from .tflite to .tfmodel) and the list of the classes in alphabetical order (.tfnames).

F. Experiment results

Performance is evaluated using a suite of standard metrics, namely accuracy, precision, recall, and F1-score. Deployment feasibility is also determined by analysing runtime memory (RAM) consumption, flash storage requirements (ROM), and the mean inference latency. To provide a realistic assessment for deployment, the following results focus exclusively on the 8-bit integer (I8) quantized models.

A binary "activity vs. no-activity" classifier is first developed to validate the pipeline and to serve as a potential low-power trigger for a multi-class model. The results demonstrate a clear advantage for the SeparableConv1D architecture. As detailed in Table XX, after 8-bit quantization, its F1-score ($\sim$85.17-98.17% on test set) are statistically comparable to the more complex Baseline CNN. Furthermore, Table XXI shows the SeparableConv1D model is significantly more efficient: 8.5x faster, with 3.7x less RAM usage and a 5.8x smaller flash footprint than the baseline.

To address the complex 19-class problem, distinguishing 18 fitness activities and a 'null' class. The SeparableConv1D model again proved optimal. As shown in Table XXII, it achieved the highest test F1-score (47.17%), outperforming the Baseline (46.33%) and the more complex Multi-branch (44.67%) and Multi-scale (44.83%) models. Notably, increased architectural complexity did not improve accuracy. Table XXIII confirms its superior efficiency, being nearly twice as fast than the baseline, making it the only practical choice. A closer look at the confusion matrix (Figure 47) shows it excels at clearly distinct activities like burpees, yet struggles to tell apart kinematically similar exercises (e.g., different jogging styles)—an inherent limitation of using a single wrist-worn sensor.

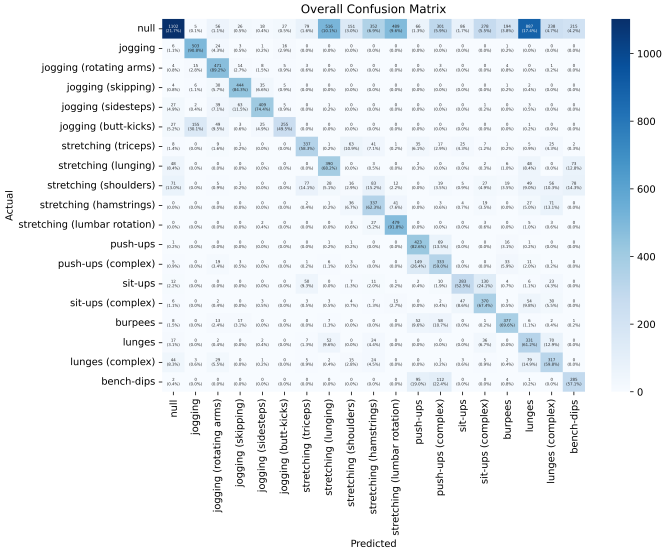

Fig. 47: Multiclass classification: Best model confusion matrix

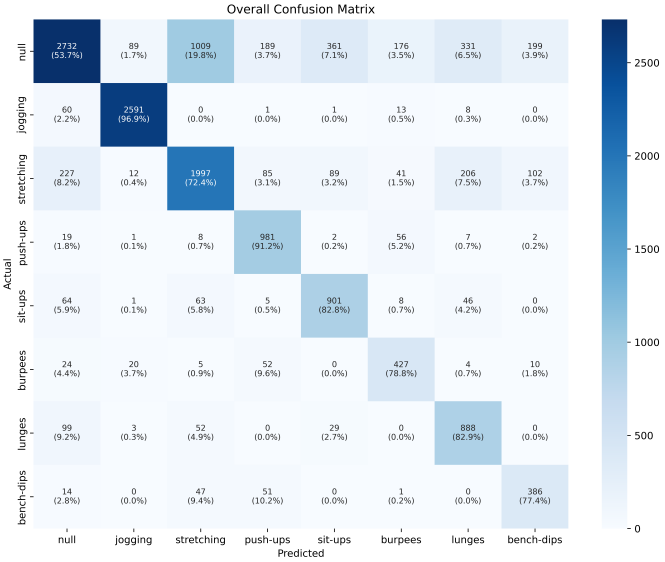

Fig. 48: Merged classification: Best model confusion matrix

The above analysis indicates a fundamental limitation in using a single wrist-worn sensor to distinguish such fine-grained movements. To create a more robust and practical model, the 18 activities were consolidated into 7 logical super-classes, with the 'null' class retained for a new total of 8, as detailed in Table XVIII.

TABLE XVIII: Consolidation of 18 activities into 7 logical super-classes. The 'null' class is retained, creating a 8-merged-class problem.

Super-class	Included Sub-classes	No. Samples
null	null	22 055
jogging	jogging, jogging with rotating arms, jogging with skipping, jogging with sidesteps, jogging with butt-kicks	8835
stretching	stretching with triceps, lunging, shoulders, hamstrings, lumbar rotation	8648
push-ups	standard and complex push-ups	3377
sit-ups	standard and complex sit-ups	3674
lunges	lunges	1734
burpees	burpees	3612
bench-dips	bench-dips	1558

Table XXIV shows a drop in both accuracy and validation but maintained high and stable F1-scores (∼68–70%), with the Multi-branch variant performing marginally best (69.67%). In terms of efficiency (Table XXV), the 8-bit quantized SeparableConv1D model was superior, achieving the lowest latency (90 ms), RAM usage (18.3 KB), and flash footprint (101.4 KB) while delivering a competitive F1-score of 69.00%. The confusion matrix (Figure 48) revealed high accuracy for distinct activities like jogging (96.9%) and push-ups (91.2%). However, significant confusion existed between the "null" and "stretching" classes, and between compound exercises and their components (e.g., burpees misclassified as push-ups). Per-subject analysis (Figure 49) confirmed these performance trends are consistent across individuals.

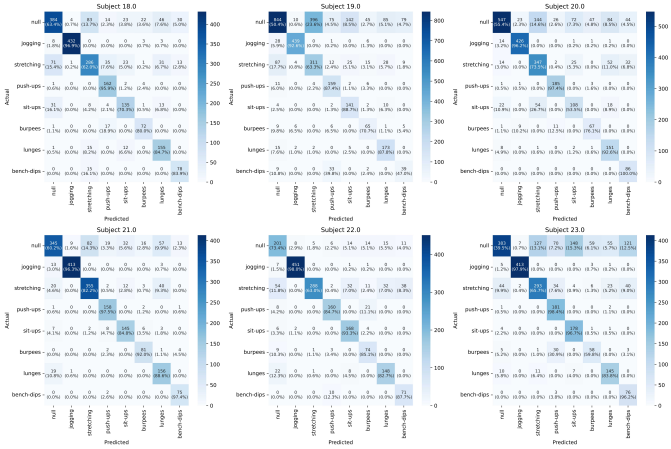

Fig. 49: Merged classification: Best model per subject confusion matrix

G. Summary

Our initial analysis of the 19-class problem revealed that while the SeparableConv1D model was the most efficient, its predictive performance was limited (F1-score of 47.17%), struggling to distinguish between kinematically similar activities. This makes the 19-class approach impractical for reliable deployment. Consequently, the activities were consolidated into 8 logical super-classes. In this merged-class scenario, the SeparableConv1D model again demonstrated superior efficiency (90ms latency, 18.3 KB RAM, and 101.4 KB ROM) while achieving a competitive F1-score of 69.00%, confirming its suitability as a robust and practical solution for resource-constrained hardware.

TABLE XIX: Performance and resource utilization for merged-class (8) classification (fold 6 of 6). F32 denotes 32-bit float; I8 denotes 8-bit integer.

Model	F32							I8						
	V_A	T_A	V_{F1}	T_{F1}	L	R	F	V_A	T_A	V_{F1}	T_{F1}	L	R	F
Baseline CNN	66.0	58.5	69	66	8408	114.0	1100.0	66.0	58.0	69	66	204	67.5	326.5
SeparableConv1D	66.3	55.4	66	69	952	30.1	191.5	62.9	50.8	63	66	90	18.3	101.4

TABLE XX: Binary classification: Comparison of model performance metrics. Each activity was treated as the positive class in a one-vs-rest scenario. A: Accuracy, P: Precision, R: Recall, and F1: F1-score, presented as mean ± standard deviation from a 6-fold cross-validation on 8-bit integer quantized models. Best results per modality are in **bold**.

Fitness Activity	Model	Validation (I8)				Test (I8)			
		V_A	V_P	V_R	V_{F1}	T_A	T_P	T_R	T_{F1}
Jogging	Baseline CNN	97.52 ± 0.99	97.33 ± 0.82	97.17 ± 0.98	97.17 ± 0.98	98.12 ± 0.17	98.17 ± 0.41	98.17 ± 0.41	98.17 ± 0.41
	SeparableConv1D	97.20 ± 1.36	97.33 ± 1.37	97.17 ± 1.47	97.17 ± 1.47	97.70 ± 0.66	98.33 ± 0.52	98.17 ± 0.75	98.17 ± 0.75
Stretching	Baseline CNN	83.97 ± 2.14	87.33 ± 1.03	83.83 ± 2.14	84.83 ± 1.60	79.42 ± 1.54	86.50 ± 0.55	84.17 ± 1.33	85.00 ± 1.26
	SeparableConv1D	84.45 ± 2.09	87.50 ± 0.84	84.50 ± 2.26	85.50 ± 1.38	80.28 ± 1.73	86.50 ± 0.55	84.50 ± 1.05	**85.17 ± 0.75**
Push-up	Baseline CNN	94.80 ± 1.11	96.33 ± 0.52	94.67 ± 1.37	95.17 ± 0.98	93.10 ± 0.99	96.00 ± 0.00	94.00 ± 0.63	94.50 ± 0.55
	SeparableConv1D	95.88 ± 0.79	96.17 ± 0.41	95.83 ± 0.98	96.00 ± 0.63	94.38 ± 0.39	95.67 ± 0.52	95.33 ± 0.52	**95.67 ± 0.52**
Sit-up	Baseline CNN	95.13 ± 0.70	96.17 ± 0.41	95.00 ± 0.89	95.33 ± 0.52	93.08 ± 1.50	95.00 ± 0.00	94.00 ± 0.89	94.50 ± 0.55
	SeparableConv1D	95.42 ± 0.98	96.00 ± 0.89	95.50 ± 1.05	95.67 ± 1.03	93.98 ± 1.16	95.50 ± 0.55	95.00 ± 0.63	**95.33 ± 0.82**
Burpees	Baseline CNN	95.73 ± 1.25	97.00 ± 0.89	95.67 ± 1.37	96.33 ± 1.03	95.18 ± 1.37	97.00 ± 0.00	96.00 ± 1.10	**96.50 ± 0.55**
	SeparableConv1D	95.10 ± 1.57	97.00 ± 0.63	94.83 ± 1.60	96.00 ± 1.26	94.23 ± 1.62	97.17 ± 0.41	95.17 ± 1.17	96.00 ± 1.10
Lunges	Baseline CNN	92.75 ± 1.96	94.67 ± 1.37	92.67 ± 1.75	93.33 ± 1.63	91.72 ± 1.52	94.33 ± 0.52	93.33 ± 1.21	93.67 ± 0.82
	SeparableConv1D	92.65 ± 1.03	94.50 ± 1.22	92.67 ± 0.82	93.33 ± 1.21	92.17 ± 0.91	94.67 ± 0.52	93.67 ± 0.82	**94.17 ± 0.41**
Bench-dips	Baseline CNN	93.48 ± 1.27	97.33 ± 0.52	93.50 ± 1.38	95.00 ± 0.89	92.22 ± 1.08	97.33 ± 0.52	93.67 ± 1.21	94.67 ± 0.82
	SeparableConv1D	95.18 ± 0.91	97.50 ± 0.55	95.00 ± 0.89	95.83 ± 0.75	94.83 ± 0.79	98.00 ± 0.00	95.67 ± 0.52	**96.33 ± 0.52**

TABLE XXI: Binary classification: Comparison of model efficiency metrics. L: inference latency; R: RAM usage; F: flash memory footprint.

Model	F32			I8		
	L (ms)	R (KB)	F (KB)	L (ms)	R (KB)	F (KB)
Baseline CNN	8068	114.0	1100.0	153	67.5	326.1
SeparableConv1D	952	30.1	190.0	85	18.3	101.0

TABLE XXII: Multi-class classification: Comparison of model performance metrics. All similar symbols are defined above.

Model	Validation (I8)				Test (I8)			
	V_A	V_P	V_R	V_{F1}	T_A	T_P	T_R	T_{F1}
Baseline CNN	44.08 ± 3.70	58.83 ± 5.49	43.83 ± 3.66	43.67 ± 5.32	30.53 ± 3.52	60.50 ± 0.84	48.83 ± 3.19	46.33 ± 4.50
SeparableConv1D	44.30 ± 3.13	60.33 ± 4.72	44.33 ± 2.88	43.67 ± 3.78	31.62 ± 1.73	61.00 ± 1.26	49.50 ± 1.05	**47.17 ± 0.98**
Multi-branch CNN	41.28 ± 3.47	57.67 ± 4.89	41.17 ± 3.43	40.50 ± 3.67	30.48 ± 0.95	59.17 ± 1.17	46.67 ± 2.94	44.67 ± 3.61
Multi-scale CNN	39.60 ± 3.29	59.67 ± 6.22	39.67 ± 3.27	38.00 ± 3.03	28.60 ± 3.15	60.67 ± 1.86	46.83 ± 0.98	44.83 ± 1.47

TABLE XXIII: Multi-class classification: Comparison of model efficiency metrics. All similar symbols are defined above.

Model	F32				I8			
	L (ms)	R (KB)	F (KB)	T_{F1} (%)	L (ms)	R (KB)	F (KB)	T_{F1} (%)
Baseline CNN	8548	114.0	1100.0	41.17 ± 4.79	146	67.5	334.6	46.33 ± 4.50
SeparableConv1D	869	30.1	194.3	45.83 ± 4.17	78	18.3	102.2	47.17 ± 0.98
Multi-branch CNN	1365	53.9	145.7	47.00 ± 3.46	293	24.9	92.5	44.67 ± 3.61
Multi-scale CNN	1936	44.8	345.8	44.83 ± 3.76	178	29.7	148.2	44.83 ± 1.47

TABLE XXIV: Merged-class classification: Comparison of model performance metrics. All similar symbols are defined above.

Model Architecture	Validation (I8)				Test (I8)			
	V_A	V_P	V_R	V_{F1}	T_A	T_P	T_R	T_{F1}
Baseline CNN	65.50 ± 2.24	69.50 ± 2.07	65.33 ± 2.16	65.50 ± 2.07	52.43 ± 4.14	70.83 ± 1.47	68.17 ± 1.72	68.00 ± 1.67
SeparableConv1D	65.70 ± 3.18	70.67 ± 1.97	65.50 ± 3.21	65.83 ± 2.56	54.90 ± 3.57	73.17 ± 1.47	69.17 ± 2.64	69.00 ± 2.53
Multi-branch CNN	66.38 ± 4.73	70.50 ± 2.88	66.50 ± 4.68	66.67 ± 4.23	54.90 ± 2.22	72.00 ± 0.89	69.67 ± 2.25	**69.67 ± 2.25**
Multi-scale CNN	65.55 ± 3.52	71.17 ±2.40	65.50 ± 3.51	65.33 ± 3.08	56.45 ± 3.28	73.33 ± 1.21	69.67 ± 2.07	69.50 ± 1.97

TABLE XXV: Merged-class classification: Comparison of model efficiency metrics. All similar symbols are defined above.

Model	F32				I8			
	L (ms)	R (KB)	F (KB)	T_{F1} (%)	L (ms)	R (KB)	F (KB)	T_{F1} (%)
Baseline CNN	8408	114.0	1100.0	65.15 $\pm$ 2.23	204	67.5	326.5	68.00 $\pm$ 1.67
SeparableConv1D	952	30.1	191.5	67.83 $\pm$ 3.11	90	18.3	101.4	69.00 $\pm$ 2.53
Multi-branch CNN	1428	53.9	142.9	71.40 $\pm$ 3.46	170	24.9	91.7	69.67 $\pm$ 2.25
Multi-scale CNN	1999	44.8	343.0	72.05 $\pm$ 3.34	175	29.7	147.5	69.50 $\pm$ 1.97

V. Conclusion

This project evaluated deep learning models for Human Activity Recognition (HAR) on the Bangle.js 2, demonstrating that the optimal architecture is highly dependent on the dataset's complexity and deployment constraints. The key findings from each case study are summarized below.

- **HANG Time-HAR:** For this dataset of complex basketball activities, the DeepConvLSTM model achieved the highest test accuracy (79.3%). This suggests that architectures capable of learning spatio-temporal features are effective for fine-grained, dynamic motions.
- **PAMAP2:** This case study highlighted a key accuracy-efficiency trade-off. While the DeepConvLSTM model achieved 90.5% accuracy, the simpler standard CNN also performed well at 85.7%, presenting a more resource-efficient alternative for deployment.
- **WEAR:** The WEAR study emphasized the need for targeted optimization. A lightweight SeparableConv1D model with 8-bit quantization proved most suitable, achieving a 69.00% F1-score on a merged-class task while providing essential efficiency gains for on-device operation.

In conclusion, the selection of an HAR model for edge devices requires a careful balance between activity complexity and hardware limitations. Future work should validate these emulator-based findings on physical hardware to confirm real-world performance and latency.

Acknowledgment

We would like to thank our supervisor Prof. Dr. Kristof Van Laerhoven for open-minded and great guidance. Thanks also go to Marius Bock from the UbiComp group for many support during the project.

References

[1] A. Ahoelzemann, "Hang Time-HAR: A Dataset for Basketball Activity Recognition," *Online*, ahoelzemann.github.io/hangtime_har, 2025.

[2] D. Alghazzawi et al., "Sensor-Based Human Activity Recognition in Smart Homes Using Depthwise Separable Convolutions," *Human-Centric Computing and Information Sciences*, vol. 12, 2022.

[3] F. Ordóñez and D. Roggen, "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition," *Sensors*, vol. 16, no. 1, 2016.

[4] R. Samanta et al., "Optimizing TinyML: The Impact of Reduced Data Acquisition Rates for Time Series Classification on Microcontrollers," *arXiv:2409.10942*, 2024.

[5] Espruino. Bangle.js 2 Technical Information. Available: https://www.espruino.com/Bangle.js2+Technical. Accessed on: Jul. 16, 2025.

[6] Espruino. Bangle.js machine learning with Edge Impulse and Tensorflow. Available: https://www.espruino.com/Bangle.js+EdgeImpulse. Accessed on: Jul. 16, 2025.

[7] M. Bock, H. Kuehne, K. Van Laerhoven, and M. Moeller, "WEAR: An Outdoor Sports Dataset for Wearable and Egocentric Activity Recognition," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 8, no. 4, pp. 1–21, Nov. 2024, doi:10.1145/3699776.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, white paper, Google, 2015. [Online]. Available: https://download.tensorflow.org/paper/whitepaper2015.pdf

[9] J. Van Der Donckt, J. Van Der Donckt, and S. Van Hoecke, "Left-Right Swapping and Upper-Lower Limb Pairing for Robust Multi-Wearable Workout Activity Detection," in *Companion of the 2024 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, Melbourne VIC, Australia: ACM, Oct. 2024, pp. 545–550, doi:10.1145/3675094.3678453.

[10] LiteRT and TensorFlow operator compatibility. Available: https://ai.google.dev/edge/litert/models/ops_compatibility. Accessed on: Jun. 20, 2025.

[11] R. Raj and A. Kos, "An improved human activity recognition technique based on convolutional neural network," *Sci Rep.*, vol. 13, no. 1, Dec. 2023, doi:10.1038/s41598-023-49739-1.

[12] N. T. H. Thu and D. S. Ha, "Depthwise separable convolution for human activity recognition," in *Proceedings of the KICS Summer Conference 2020 Program*, 2020.

[13] S. Zhang *et al.*, "Deep Learning in Human Activity Recognition with Wearable Sensors: A Review on Advances," *Sensors*, vol. 22, no. 4, p. 1476, Feb. 2022, doi:10.3390/s22041476.

[14] D. Alghazzawi, O. Rabie, O. Bamasaq, A. Albeshri, and M. Z. Asghar, "Sensor-Based Human Activity Recognition in Smart Homes Using Depthwise Separable Convolutions," *Human-centric Computing and Information Sciences*, vol. 12, no. 0, pp. 676–694, Oct. 2022, doi:10.22967/HCIS.2022.12.050.

[15] Model Optimization. Available: https://ai.google.dev/edge/litert/models/model_optimization. Accessed on: Jun. 20, 2025.

[16] Post Training Quantization. Available: https://ai.google.dev/edge/litert/models/post_training_quantization. Accessed on: Jun. 20, 2025.

[17] A. Calev, "Time Series Models PAMAP2 Dataset," *Kaggle*. Accessed: Jul. 18, 2025. [Online]. Available: https://www.kaggle.com/code/avrahamcalev/time-series-models-pamap2-dataset/notebook

[18] Espruino. Espruino IDE. Available: https://www.espruino.com/IDE. Accessed on: Jun. 20, 2025.

[19] Espruino. Bangle.js 2. Available: https://www.espruino.com/Bangle.js2. Accessed on: Jun. 20, 2025.

Supplementary Materials

Emin Abdurahmanov, Saju Khakurel, and Thong Phan

I. PAMAP2

A. Model Architecture

To maintain consistency and allow for direct comparison with the other case studies, we evaluated four distinct neural network architectures. All models were designed to process an input shape of (100, 17), corresponding to the 2-second data windows with 17 features, and to classify the 7 selected activities. The specifics of each architecture are detailed below.

TABLE I: LSTM Architecture. This model uses two stacked LSTM layers to capture temporal dependencies.

Layer (type)	**Output Shape**	**Param #**
lstm (LSTM)	(None, 100, 64)	20,992
dropout (Dropout)	(None, 100, 64)	0
lstm_1 (LSTM)	(None, 64)	33,024
dropout_1 (Dropout)	(None, 64)	0
dense (Dense)	(None, 7)	455
Total params:		**54,471**
Trainable params:		54,471
Non-trainable params:		0

TABLE II: CNN Architecture. This model employs 1D convolutional layers to extract spatial features.

Layer (type)	**Output Shape**	**Param #**
Conv1D	(None, 98, 64)	3,328
MaxPooling1D	(None, 49, 64)	0
Conv1D	(None, 47, 128)	24,704
MaxPooling1D	(None, 23, 128)	0
Flatten	(None, 2,944)	0
Dense	(None, 100)	294,500
Dropout	(None, 100)	0
Dense	(None, 7)	707
Total params:		**323,239**
Trainable params:		323,239
Non-trainable params:		0

TABLE III: Hybrid CNN-LSTM Architecture. This architecture combines a convolutional layer with LSTM layers.

Layer (type)	**Output Shape**	**Param #**
Conv1D	(None, 98, 64)	3,328
MaxPooling1D	(None, 49, 64)	0
Reshape	(None, 49, 64)	0
LSTM	(None, 49, 64)	33,024
Dropout	(None, 49, 64)	0
LSTM	(None, 64)	33,024
Dropout	(None, 64)	0
Dense	(None, 7)	455
Total params:		**69,831**
Trainable params:		69,831
Non-trainable params:		0

TABLE IV: DeepConvLSTM Model Architecture. This is the most complex model, with multiple convolutional and LSTM layers.

Layer (type)	**Output Shape**	**Param #**
Conv1D	(None, 100, 64)	5,504
Dropout	(None, 100, 64)	0
Conv1D	(None, 100, 64)	20,544
Dropout	(None, 100, 64)	0
Conv1D	(None, 100, 64)	20,544
Dropout	(None, 100, 64)	0
Conv1D	(None, 100, 64)	20,544
Dropout	(None, 100, 64)	0
LSTM	(None, 100, 128)	98,816
Dropout	(None, 100, 128)	0
LSTM	(None, 128)	131,584
Dropout	(None, 128)	0
Dense	(None, 7)	903
Total params:		**298,439**
Trainable params:		298,439
Non-trainable params:		0

II. WEAR

A. Pre-processing Techniques

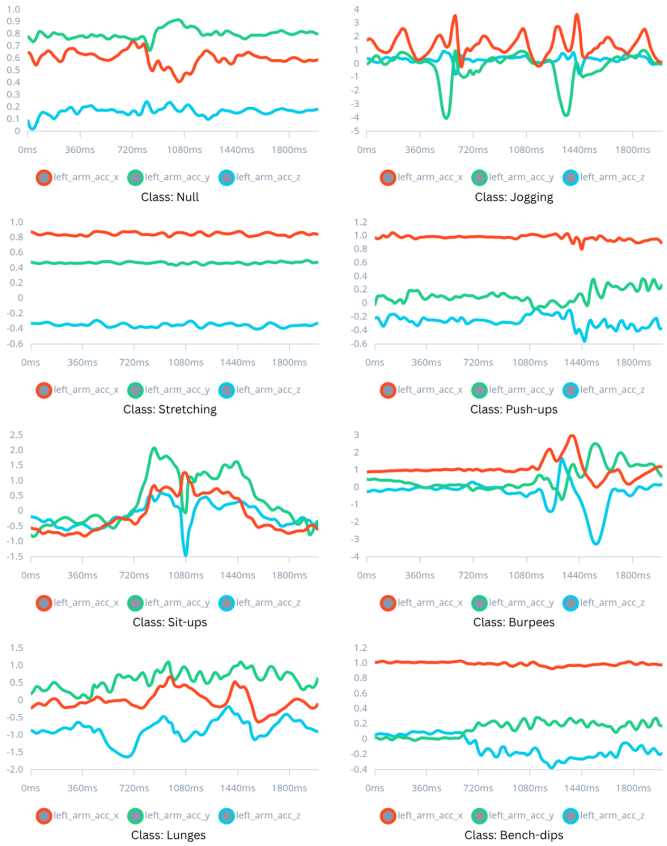

Fig. 1: Visualization of raw 3-axis accelerometer data from the left wrist sensor for a selection of representative activities. Each subplot shows a 2-second time-series window, highlighting the distinct temporal patterns and amplitude variations between different classes.

TABLE V: Statistics for the 3-axis accelerometer data from the left wrist sensor across the entire WEAR dataset.

Metrics	**left_arm_acc_x**	**left_arm_acc_y**	**left_arm_acc_z**
mean	$-4.259\,147 \times 10^{-1}$	$3.683\,402 \times 10^{-1}$	$-2.195\,465 \times 10^{-1}$
std	$8.065\,931 \times 10^{-1}$	$6.518\,021 \times 10^{-1}$	$4.909\,763 \times 10^{-1}$
min	$-7.993\,350$	$-5.644\,781$	$-5.275\,994$
25%	$-9.280\,867 \times 10^{-1}$	$3.349\,875 \times 10^{-2}$	$-5.191\,814 \times 10^{-1}$
50%	$-5.519\,283 \times 10^{-1}$	$3.244\,143 \times 10^{-1}$	$-2.184\,292 \times 10^{-1}$
75%	$1.042\,793 \times 10^{-1}$	$6.675\,899 \times 10^{-1}$	$3.636\,829 \times 10^{-2}$
max	$7.992\,366$	$7.995\,070$	$5.023\,094$

B. Model Architecture

To initiate our experimental evaluation, we employ a foundational 7-layer CNN. Each convolutional layer learns to extract local temporal patterns by convolving its filters over successive time-steps, enabling the network to capture dependencies between adjacent samples in the time-series data. This inherent ability to model temporal structure makes CNNs particularly well-suited for human activity recognition from inertial data. The architecture consists of seven convolutional-pooling blocks: an initial block with 256 filters with small kernel size (3), three subsequent blocks with 128 filters with the same kernel size (3), and three final bottleneck blocks with 64 filters with smallest kernel size (1). After the convolutional layers, the feature maps are flattened and passed to a classifier head composed of three dense layers (256, 128, and 64 units) regularized with 40% dropout and L_1 ($\lambda = 10^{-5}$). A final softmax layer produces the class probabilities.

TABLE VI: Baseline CNN Architecture.

Layer (type)	**Output Shape**	**Param #**
Reshape	(None, 100, 3)	0
Conv1D (256 filters)	(None, 100, 256)	2,560
MaxPooling1D	(None, 50, 256)	0
Conv1D (128 filters)	(None, 50, 128)	98,432
MaxPooling1D	(None, 25, 128)	0
Conv1D (128 filters)	(None, 25, 128)	49,280
MaxPooling1D	(None, 13, 128)	0
Conv1D (128 filters)	(None, 13, 128)	49,280
MaxPooling1D	(None, 7, 128)	0
Conv1D (64 filters)	(None, 7, 64)	8,256
MaxPooling1D	(None, 4, 64)	0
Conv1D (64 filters)	(None, 4, 64)	4,160
MaxPooling1D	(None, 2, 64)	0
Conv1D (64 filters)	(None, 2, 64)	4,160
MaxPooling1D	(None, 1, 64)	0
Flatten	(None, 64)	0
Dense (256 units)	(None, 256)	16,640
Dense (128 units)	(None, 128)	32,896
Dense (64 units)	(None, 64)	8,256
Dropout (0.4)	(None, 64)	0
y_pred (Dense, 8 classes)	(None, 8)	520
Total params: 274,440		
Trainable params: 274,440		
Non-trainable params: 0		

Although the standard-convolutional network effectively captured temporal features, its substantial parameter count resulted in excessive memory consumption on our resource-constrained dataset. To address these limitations, the baseline was redesigned using SeparableConv1D and GlobalAveragePooling. The revised architecture comprises a SeparableConv1D stem with bigger kernel size (5), followed by three separable blocks ($64 \rightarrow 128 \rightarrow 128$ channels) with smaller kernel size (3). A GAP layer replaces the baseline's flatten and dense stack, feeding directly into a final regularized dense layer (64 units, L_2) and softmax output.

TABLE VII: SeparableConv1D Architecture.

Layer (type)	**Output Shape**	**Param #**
Reshape	(None, 100, 3)	0
BatchNormalization	(None, 100, 3)	12
SeparableConv1D (32 filters, k=5)	(None, 100, 32)	143
BatchNormalization	(None, 100, 32)	128
MaxPooling1D (pool=2)	(None, 50, 32)	0
SeparableConv1D (64 filters, k=3)	(None, 50, 64)	2,208
BatchNormalization	(None, 50, 64)	256
MaxPooling1D (pool=2)	(None, 25, 64)	0
SeparableConv1D (128 filters, k=3)	(None, 25, 128)	8,512
BatchNormalization	(None, 25, 128)	512
MaxPooling1D (pool=2)	(None, 13, 128)	0
SeparableConv1D (128 filters, k=3)	(None, 13, 128)	16,896
BatchNormalization	(None, 13, 128)	512
MaxPooling1D (pool=2)	(None, 7, 128)	0
GlobalAveragePooling1D	(None, 128)	0
Dropout (rate=0.4)	(None, 128)	0
Dense (64 units, $L_2 = 10^{-4}$)	(None, 64)	8,256
y_pred (Dense, 8 classes)	(None, 8)	520
Total params: 37,955		
Trainable params: 37,245		
Non-trainable params: 710		

To explore a more sophisticated feature extraction strategy, we introduce a dual-branch architecture. The model is designed to capture multi-scale temporal features by processing the input in parallel before fusing the information. After reshaping the flat input into a (100, 3) time-channel tensor, two parallel branches independently process the data. The first branch intends to capture fine-grained features with small kernels (3) learn local, high-resolution temporal dependencies. In contrast, the second branch aims to capture coarse features by using bigger kernels size (7 and 5). Both branches include a batch normalization after each convolution to stabilize training, and a pooling layer to halve the sequence length.

TABLE VIII: Multi-branch CNN Architecture.

Layer (type)	**Output Shape**	**Param #**
input	(None, 300)	0
Reshape	(None, 100, 3)	0
Branch 1 (fine-grained)		
SeparableConv1D (32 filters, k=3)	(None, 100, 32)	137
BatchNormalization	(None, 100, 32)	12
SeparableConv1D (48 filters, k=3)	(None, 100, 48)	1,680
BatchNormalization	(None, 100, 48)	192
MaxPooling1D (pool_size=2)	(None, 50, 48)	0
Branch 2 (coarse)		
SeparableConv1D (32 filters, k=7)	(None, 100, 32)	149
BatchNormalization	(None, 100, 32)	128
SeparableConv1D (48 filters, k=5)	(None, 100, 48)	1,744
BatchNormalization	(None, 100, 48)	192
MaxPooling1D (pool_size=2)	(None, 50, 48)	0
Merge + further processing		
Concatenate	(None, 50, 96)	0
SeparableConv1D (64 filters, k=3)	(None, 50, 64)	6,496
BatchNormalization	(None, 50, 64)	256
MaxPooling1D (pool_size=2)	(None, 25, 64)	0
SeparableConv1D (96 filters, k=3)	(None, 25, 96)	6,432
BatchNormalization	(None, 25, 96)	384
GlobalAveragePooling1D	(None, 96)	0
Classification head		
Dense (64 units, $L_2 = 10^{-4}$)	(None, 64)	6,208
Dropout (rate=0.4)	(None, 64)	0
y_pred (Dense 8-way softmax)	(None, 8)	520
Total params: 24,530 Trainable params: 23,948 Non-trainable params: 582		

To establish an upper performance benchmark, our final model employs a multi-scale architecture. It is designed to maximize accuracy by using a cascade of SeparableConv1D layers with progressively smaller kernel sizes to capture features at different temporal resolutions. It features a cascade of SeparableConv1D layers with decreasing kernel sizes (from 7 to 5, then to 3). Filter depth increases from 64 to 96, and a 1×1 convolution precedes the classifier. The head uses a GAP layer and two regularized dense layers ($128 \rightarrow 64$ units) with a final softmax output.

TABLE IX: Multi-scale CNN Architecture.

Layer (type)	**Output Shape**	**Param #**
Reshape	(None, 100, 3)	0
BatchNormalization	(None, 100, 3)	12
SeparableConv1D (32×7)	(None, 100, 32)	149
BatchNormalization	(None, 100, 32)	128
SeparableConv1D (32×5)	(None, 100, 32)	1,216
BatchNormalization	(None, 100, 32)	128
SeparableConv1D (32×3)	(None, 100, 32)	1,152
BatchNormalization	(None, 100, 32)	128
MaxPooling1D (2)	(None, 50, 32)	0
SeparableConv1D (64×5)	(None, 50, 64)	2,272
BatchNormalization	(None, 50, 64)	256
Dropout (0.2)	(None, 50, 64)	0
SeparableConv1D (96×3)	(None, 50, 96)	6,432
BatchNormalization	(None, 50, 96)	384
MaxPooling1D (2)	(None, 25, 96)	0
SeparableConv1D (128×3)	(None, 25, 128)	12,704
BatchNormalization	(None, 25, 128)	512
Dropout (0.3)	(None, 25, 128)	0
SeparableConv1D (128×3)	(None, 25, 128)	16,896
BatchNormalization	(None, 25, 128)	512
MaxPooling1D (2)	(None, 13, 128)	0
SeparableConv1D (96×1)	(None, 13, 96)	12,512
BatchNormalization	(None, 13, 96)	384
GlobalAveragePooling1D	(None, 96)	0
Dropout (0.5)	(None, 96)	0
Dense (128, $L_2 = 10^{-4}$)	(None, 128)	12,416
Dropout (0.4)	(None, 128)	0
Dense (64, $L_2 = 10^{-4}$)	(None, 64)	8,256
Dropout (0.3)	(None, 64)	0
y_pred (Dense, 8 classes)	(None, 8)	520
Total params: 76,969 Trainable params: 76,358 Non-trainable params: 611		

C. Hyperparameters and Methodology

For the WEAR dataset of 24 participants, we adapt LOSO into a six-fold design as below.

TABLE X: Detail explanation of LOSO cross validation scheme in the context of the WEAR dataset.

Fold No.	Training set (15 subjects)	Validation set (3 subjects)	Test set (6 subjects)
1	3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17	0; 1; 2	18; 19; 20; 21; 22; 23
2	0; 1; 2; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17	3; 4; 5	
3	0; 1; 2; 3; 4; 5; 9; 10; 11; 12; 13; 14; 15; 16; 17	6; 7; 8	
4	0; 1; 2; 3; 4; 5; 6; 7; 8; 12; 13; 14; 15; 16; 17	9; 10; 11	
5	0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 15; 16; 17	12; 13; 14	
6	0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14	15; 16; 17	

D. Unquantized experiment results in float32

TABLE XI: Binary classification performance. A: Accuracy, P: Precision, R: Recall, and F1: F1-score, presented as mean ± standard deviation from a 6-fold cross-validation on float-32bit unquantized models.

Activity	**Model**	**Validation (F32)**				**Test (F32)**			
		V_A	V_P	V_R	V_F1	T_A	T_P	T_R	T_F1
Jogging	Baseline CNN	95.83 ± 1.23	97.00 ± 0.89	95.67 ± 1.37	96.33 ± 1.03	95.23 ± 1.39	97.00 ± 0.00	96.00 ± 1.10	96.33 ± 0.82
	SeparableConv1D	95.42 ± 1.11	97.17 ± 0.75	95.33 ± 1.03	96.17 ± 0.75	94.75 ± 0.77	97.17 ± 0.41	95.50 ± 0.55	96.50 ± 0.55
Stretching	Baseline CNN	83.33 ± 2.22	87.17 ± 1.17	83.17 ± 2.32	84.67 ± 1.86	79.37 ± 1.65	86.50 ± 0.55	83.67 ± 1.63	84.67 ± 1.03
	SeparableConv1D	83.43 ± 2.11	87.67 ± 0.82	83.67 ± 2.07	84.83 ± 1.17	79.72 ± 1.18	87.00 ± 0.63	83.50 ± 1.05	84.67 ± 1.03
Push-up	Baseline CNN	92.88 ± 2.21	95.00 ± 1.26	92.83 ± 2.14	93.50 ± 1.76	92.33 ± 1.57	94.50 ± 0.55	93.50 ± 1.22	93.67 ± 0.82
	SeparableConv1D	93.02 ± 1.51	94.50 ± 1.22	93.00 ± 1.55	93.50 ± 1.38	93.13 ± 0.51	94.83 ± 0.41	94.33 ± 0.52	94.50 ± 0.55
Sit-up	Baseline CNN	97.52 ± 0.99	97.33 ± 0.82	97.17 ± 0.98	97.17 ± 0.98	98.10 ± 0.23	98.33 ± 0.52	98.33 ± 0.52	98.33 ± 0.52
	SeparableConv1D	97.32 ± 1.31	97.50 ± 1.05	97.17 ± 1.47	97.17 ± 1.47	97.90 ± 0.61	98.50 ± 0.55	98.33 ± 0.52	98.50 ± 0.55
Burpees	Baseline CNN	94.83 ± 1.21	96.50 ± 0.55	94.83 ± 1.47	95.17 ± 0.98	93.20 ± 1.01	96.00 ± 0.00	94.00 ± 0.63	94.67 ± 0.52
	SeparableConv1D	96.03 ± 0.72	97.00 ± 0.00	96.17 ± 0.75	96.33 ± 0.52	94.60 ± 0.18	97.00 ± 0.00	95.00 ± 0.00	95.83 ± 0.41
Lunges	Baseline CNN	95.35 ± 0.65	96.17 ± 0.41	95.33 ± 0.82	95.67 ± 0.52	93.43 ± 1.39	95.17 ± 0.41	94.33 ± 1.03	94.50 ± 0.84
	SeparableConv1D	95.28 ± 0.83	96.17 ± 0.75	95.33 ± 1.03	95.67 ± 1.03	93.87 ± 0.81	95.67 ± 0.52	94.67 ± 0.52	95.00 ± 0.63
Bench-dips	Baseline CNN	93.78 ± 1.82	97.33 ± 0.52	93.67 ± 1.63	95.17 ± 1.17	92.82 ± 0.82	97.50 ± 0.55	93.67 ± 0.82	95.00 ± 0.89
	SeparableConv1D	95.35 ± 0.95	97.67 ± 0.52	95.17 ± 0.75	96.17 ± 0.75	95.47 ± 0.50	98.00 ± 0.00	96.33 ± 0.52	96.67 ± 0.52

TABLE XII: Multi-class classification performance. A: Accuracy, P: Precision, R: Recall, and F1: F1-score, presented as mean ± standard deviation from a 6-fold cross-validation on 8-bit integer quantized models.

Model	**Validation (F32)**				**Test (F32)**			
	V_A	V_P	V_R	V_F1	T_A	T_P	T_R	T_F1
Baseline CNN	42.45 ± 3.55	59.33 ± 5.47	42.33 ± 3.39	41.17 ± 4.79	30.62 ± 3.54	60.83 ± 0.75	47.00 ± 2.83	43.67 ± 4.27
SeparableConv1D	46.58 ± 3.60	62.67 ± 5.50	46.67 ± 3.67	45.83 ± 4.17	35.23 ± 2.04	63.33 ± 1.21	51.83 ± 2.14	49.17 ± 2.71
Multi-branch CNN	47.38 ± 3.24	63.00 ± 4.69	47.33 ± 3.27	47.00 ± 3.46	34.97 ± 1.45	64.00 ± 0.63	53.17 ± 1.17	51.83 ± 1.47
Multi-scale CNN	45.87 ± 3.70	63.83 ± 4.88	46.00 ± 3.74	44.83 ± 3.76	33.83 ± 3.28	66.33 ± 1.21	54.33 ± 1.63	53.00 ± 2.00

TABLE XIII: Merged-class classification performance. A: Accuracy, P: Precision, R: Recall, and F1: F1-score, presented as mean ± standard deviation from a 6-fold cross-validation on 8-bit integer quantized models.

Model	**Validation (F32)**				**Test (F32)**			
	V_A	V_P	V_R	V_F1	T_A	T_P	T_R	T_F1
Baseline CNN	65.15 ± 2.23	69.83 ± 1.94	65.17 ± 2.32	65.17 ± 2.14	52.77 ± 4.26	70.83 ± 1.47	67.67 ± 1.75	65.15 ± 2.23
SeparableConv1D	67.83 ± 3.11	72.00 ± 2.37	67.83 ± 3.19	67.83 ± 2.86	58.15 ± 2.44	74.50 ± 1.22	71.50 ± 1.38	67.83 ± 3.11
Multi-branch CNN	71.40 ± 3.46	73.83 ± 3.13	71.33 ± 3.50	71.33 ± 3.61	60.55 ± 2.50	75.17 ± 0.75	73.50 ± 0.84	71.40 ± 3.46
Multi-scale CNN	72.05 ± 3.34	75.17 ± 2.99	72.17 ± 3.37	72.00 ± 3.58	62.38 ± 3.08	77.00 ± 0.89	75.33 ± 1.03	72.05 ± 3.34