ASEAN Engineering Journal

Full Paper

FPGA-BASED MULTILAYER PERCEPTRON FOR FAST HUMAN ACTIVITY DETECTION

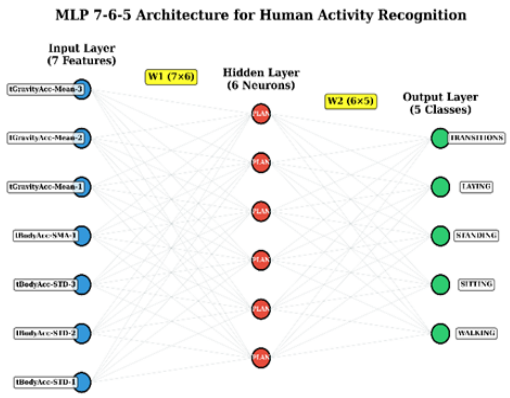
M Mohammed Sabir Hussain[a*], Taha Mazher[a], M A Raheem[a], Hakeem Ajaz Aslam[a], K Sasidhar[a], Afaq Ahmed[b]

[a]Department of Electronics & Communication Engineering, Muffakham Jah College of Engineering & Technology, India.
[b]Modern College of Business and Sciences, Bawshar St, Muscat 133, Oman

Article history
Received
17 July 2025
Received in revised form
16 October2025
Accepted
30 October 2025
Published online
31 May 2026

*Corresponding author
sabirhussain@mjcollege.ac.in

Graphical abstract



Abstract

This research presents an efficient Field-Programmable Gate Array (FPGA) implementation of a 7-6-5 Multi-Layer Perceptron (MLP) neural network for human activity recognition (HAR) using the UCI HAR dataset. The study addresses the growing need for low-power, real-time activity recognition systems suitable for embedded and wearable applications. A novel ReLU6 activation function is adopted to optimize the neural network for hardware deployment. The system processes seven selected features from accelerometer and gyroscope data to classify five activity classes: Walking, Sitting, Standing, Laying, and Activity Transitions. The design targets the Xilinx Artix-7 35T FPGA and includes complete Verilog hardware generation, encompassing network parameters, RELU6 activation lookup tables, and testbench validation. Quantitative results show high classification accuracy: 94.4% on Dataset A, 94.7% on Dataset B, and 94.1% on Dataset C, demonstrating consistent performance across different data splits. The results confirm that the proposed FPGA-based MLP offers a complete, real-time, hardware-ready solution for human activity recognition.

Keywords: FPGA, Human Activity Recognition (HAR), Multi-Layer Perceptron (MLP), RELU6 Activation, UCI HAR Dataset, Embedded Systems, Real-time Processing.

1.0 INTRODUCTION

Human Activity Recognition (HAR) has become a key technology in intelligent systems, underpinning applications in healthcare, assisted living, sports monitoring, and smart environments [11], [12]. The widespread adoption of smartphones and wearable devices equipped with inertial sensors has enabled continuous monitoring of human motion, generating large amounts of data that can be used to infer user context and behaviour. Accurate recognition of such activities in real time remains a critical challenge, particularly for resource-limited embedded devices.

Conventional HAR approaches employing machine learning models such as Support Vector Machines and Random Forests are typically executed on general-purpose processors. Although these methods achieve satisfactory accuracy, they are not well suited for embedded deployment due to high computational overhead, latency, and energy consumption. Deep learning models, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have further improved accuracy but at the cost of significantly increased hardware complexity and memory requirements. These characteristics hinder their implementation in wearable or portable systems that demand continuous, low-power operation.

Recent advances in Field-Programmable Gate Arrays (FPGAs) offer an attractive alternative for edge-level intelligence by enabling parallel processing, deterministic timing, and power-efficient computation [1], [2], [3]. Nevertheless, many existing FPGA-based HAR solutions continue to rely on complex network topologies and non-linear activation functions that require floating-point arithmetic, resulting in high logic utilization and limited scalability. This indicates a persisting gap between software-level accuracy and hardware-level efficiency.

In this work, an efficient FPGA-based Multilayer Perceptron (MLP) is proposed for real-time human activity detection. The network adopts a compact 7-6-5 architecture and employs the ReLU6 activation function to achieve hardware-friendly nonlinearity without exponential computations. Using a subset of seven discriminative features from the UCI HAR dataset [8],

[10] the model achieves consistent classification accuracy exceeding 94% across multiple data splits. The design is implemented on a Xilinx Artix-7 FPGA, demonstrating only 2.5% LUT utilization and a total power consumption of 67 mW. These results establish a lightweight, low-power, and hardware-optimized solution for human activity recognition, addressing the gap between high-performance neural models and deployable embedded architectures.

2.0 METHODOLOGY

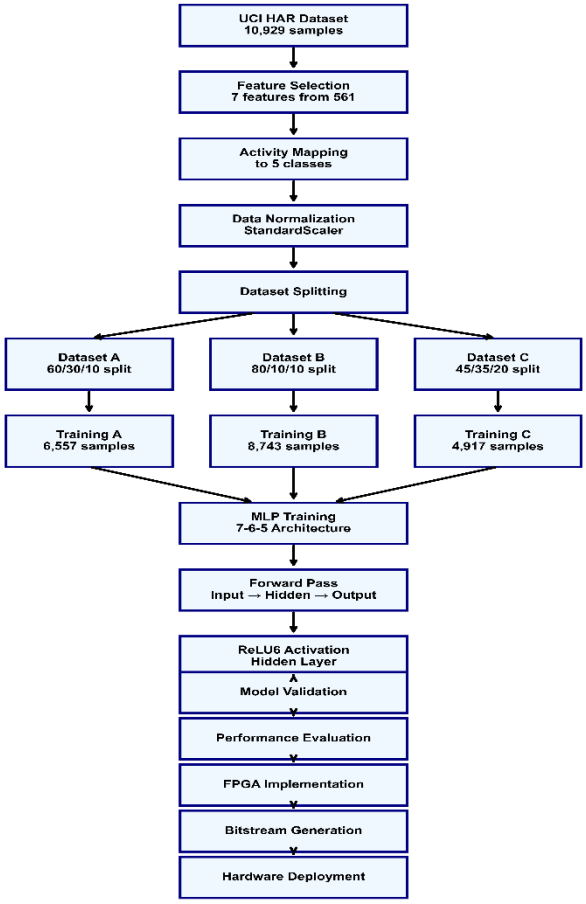


Figure 1 Complete Flowchart

The process begins with the UCI Human Activity Recognition (HAR) dataset containing 10,929 smartphone sensor samples. From the original 561 features, only 7 key features are selected to reduce dimensionality and focus on essential motion characteristics. The 6 original activity classes are consolidated into 5 classes for simplicity. Data is then normalized using StandardScaler to ensure zero mean and unit variance, followed by dataset splitting into training, validation, and test sets to create three variations—Dataset A (60/30/10), Dataset B (80/10/10), and Dataset C (45/35/20)—for robustness testing. A Multi-Layer Perceptron (MLP) with a 7-6-5 architecture is trained on each dataset using ReLU6 activation in the hidden layer for FPGA efficiency. After validation and performance evaluation (accuracy, confusion matrix, and resource use), the optimized MLP is converted to HDL, synthesized, and implemented on an FPGA board through bitstream generation and hardware deployment for real-time activity recognition.

2.1 Data Preprocessing

This study utilizes the publicly available UCI HAR dataset, which contains inertial sensor data collected from smartphones worn on the waist of 30 human subjects [8]. The dataset includes three-axis linear acceleration and three-axis angular velocity readings, sampled at 50 Hz, as participants performed various physical activities including static postures and transitional movements.

For this work, we selected five activity classes: Walking, Sitting, Standing, Laying, and a unified “Transitions” class. The latter consolidates six postural changes (e.g., sit-to-stand, lie-to-sit) into a single category to reduce label sparsity and simplify classification. This reorganization results in a balanced five-class classification problem suitable for real-time embedded implementation.

The raw time-series data was segmented using a sliding window of 2.56 seconds (128 samples), with 50% overlap between consecutive windows [23]. From each window, time- and frequency-domain features were extracted [4]. Although the original dataset includes 561 features per window, only 7 key features were selected based on domain relevance and suitability for hardware implementation. These features were:

- Body acceleration standard deviation (X-axis, Y-axis and Z-axis)
- Signal magnitude area of body acceleration
- Gravity acceleration mean (X-axis)
- Gravity acceleration mean (Y-axis)
- Gravity acceleration mean (Z-axis)

These features capture both the intensity and direction of movement, as well as orientation relative to gravity—critical cues for differentiating between physical states. All features were standardized to have zero mean and unit variance to ensure consistency across samples and improve training convergence [28].

Three dataset configurations were defined using different training, validation, and testing splits to evaluate model robustness. Details of the splits are presented in Table 1.

Table 1 Details of Classification Datasets

Dataset	Training Samples	Validation Samples	Testing Samples	Split Ratio
Dataset A	6,010	3,279	1,640	55:30:15
Dataset B	8,743	1,093	1,093	80:10:10
Dataset C	4,917	3,826	2,186	45:35:20

2.2 Neural Network Training

2.2.1 Multi-Layer Perceptron Architecture Design

The designed MLP model adopts a 7-6-5 layer structure, chosen for its balance between simplicity and performance in real-time, hardware-limited systems. It comprises 7 input neurons representing selected features, 6 hidden neurons for intermediate processing, and 5 output neurons, each corresponding to one of the recognized activities—Walking, Sitting, Standing, Laying, and Transitions [2]. In this network, every neuron calculates a weighted combination of inputs, incorporates a bias, and then applies a nonlinear activation function to produce its output [28]. The process can be mathematically expressed as:

$$a^{(l)} = f\left(W^{(l)}x^{(l-1)} + b^{(l)}\right) \quad (1)$$

Where,

- $a^{(l)}$ is the output of layer l
- $W^{(l)}$Is the weight matrix connecting layer $l-1$ to layer l
- $x^{(l-1)}$Is the input vector from the previous layer
- $b^{(l)}$ is the bias vector for layer l
- $f(\cdot)$ is the activation function implemented using RELU6

2.2.2 ReLU6 Activation Function Implementation

To implement the non-linear transformation required by the MLP, we employ the ReLU6 activation function, a bounded variant of the Rectified Linear Unit (ReLU) [6]. ReLU6 approximates non-linear behavior by clipping inputs to the range [0, 6] and normalizing the output, providing a hardware-efficient alternative to more complex activations like sigmoid or the original PLAN. This approach minimizes computational overhead, making it highly suitable for FPGA implementation with reduced resource requirements compared to piecewise linear methods.

The ReLU6 function is defined as:

$$ReLU6(x) = min\left((0, x)\right, 6)$$

1. The ReLU6 function offers several advantages over traditional activation functions, including the original PLAN:
2. Hardware Efficiency: Uses simple clipping and division operations, eliminating the need for lookup tables or piecewise segments.
3. Memory Optimization: Requires minimal logic gates, reducing FPGA resource usage (e.g., fewer LUTs and DSPs).
4. Precision Maintenance: Maintains numerical stability in fixed-point arithmetic with bounded outputs in [0, 1].
5. Gradient Preservation: Provides non-zero gradients in the active range (0 < x < 6), supporting effective backpropagation.

2.2.3 Training Algorithm and Optimization Strategy

The Multilayer Perceptron was trained using the backpropagation algorithm in combination with the Adam optimizer [28]. The network's parameters — weights and biases — were updated iteratively to minimize the loss function computed between the network's output and the true class labels.

During the forward propagation phase, the input features move sequentially through the hidden and output layers. Each neuron in the hidden layer performs a weighted combination of input values, adds a bias, and then applies the ReLU6 activation function to introduce nonlinearity. The resulting activations are transmitted to the output layer, where a linear operation produces the final class scores. The activity class with the highest output score is identified as the model's prediction.

To guide learning, the Mean Squared Error (MSE) was selected as the loss function. This choice was driven by its compatibility with the RELU6 activation function, as MSE penalizes squared deviations and performs well when outputs are continuous-valued rather than probability-normalized. The Adam optimizer was employed due to its adaptive learning rate properties, which help achieve faster convergence and more stable training. A constant learning rate of 0.001 was used across all training runs. Weights and biases in the network were initialized using the Xavier Normal initialization method to maintain variance consistency across layers, which is critical for effective gradient-based training [28]. The network was trained using a mini-batch size of 32, which provided a good trade-off between convergence speed and stability in gradient estimation [25].

To mitigate overfitting, the early stopping technique was utilized with a patience value of 50 epochs [27]. This mechanism halted training when the validation loss showed no improvement for 50 consecutive epochs, ensuring that the optimal model weights corresponding to the best validation performance were preserved. Although the training process allowed up to 500 epochs, convergence was generally observed between 140 and 260 epochs, depending on the specific dataset configuration.

Each of the three dataset splits (A, B, and C) was trained independently using the same training algorithm and hyperparameter setup. Additionally, each configuration was trained five times using different random seeds to ensure statistical consistency and robustness of results [26].

2.2.4 Training procedure and Validation Strategy

The training procedure was designed to evaluate the robustness of the MLP architecture across different data distribution scenarios. Three distinct dataset configurations (A, B, C) were employed, each with different training/validation/testing proportions to assess the model's performance under varying data availability conditions.

Dataset A Training Procedure:

- Initialization Phase: Weights initialized using Xavier normal distribution
- Training Phase: 6,010 samples used for gradient computation and weight updates
- Validation Phase: 3,279 samples used for performance monitoring and early stopping

- Testing Phase: 1,640 samples reserved for final performance evaluation

Dataset B Training Procedure:

- Initialization Phase: Identical weight initialization strategy
- Training Phase: 8,743 samples providing extensive training data
- Validation Phase: 1,093 samples for overfitting prevention
- Testing Phase: 1,093 samples for performance assessment

Dataset C Training Procedure:

- Initialization Phase: Consistent initialization across all experiments
- Training Phase: 4,917 samples representing limited training scenario
- Validation Phase: 3,826 samples for comprehensive validation
- Testing Phase: 2,186 samples for robust testing evaluation

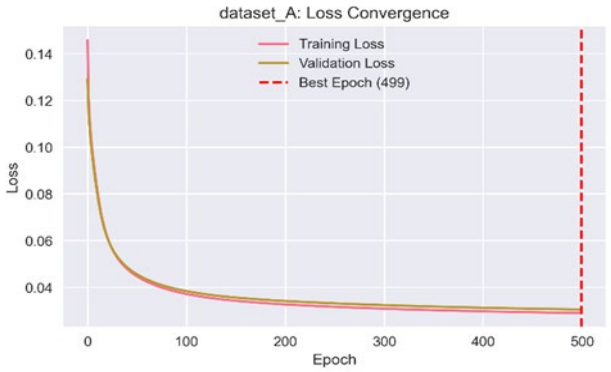
The training process was monitored using multiple metrics to ensure optimal convergence and prevent overfitting:

Primary Metrics:

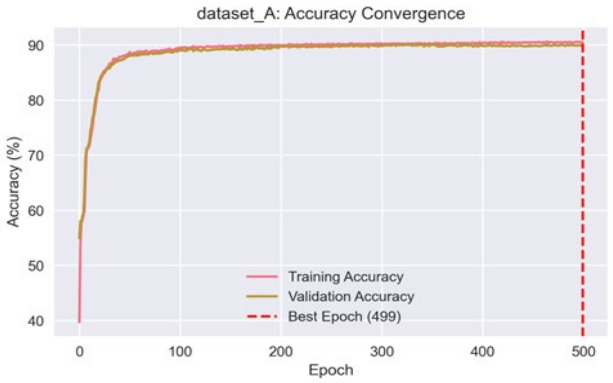
- Training Loss: MSE computed on training batches
- Validation Loss: MSE computed on validation set after each epoch
- Training Accuracy: Classification accuracy on training data
- Validation Accuracy: Classification accuracy on validation data

Secondary Metrics:

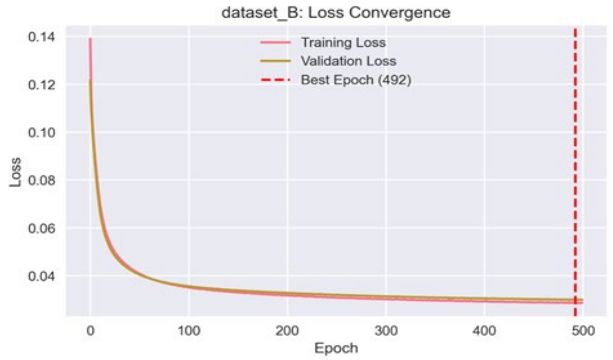
- Gradient Norm: L2 norm of gradients for convergence monitoring
- Weight Magnitude: Average weight magnitudes for stability assessment
- Learning Rate Schedule: Adaptive learning rate adjustments



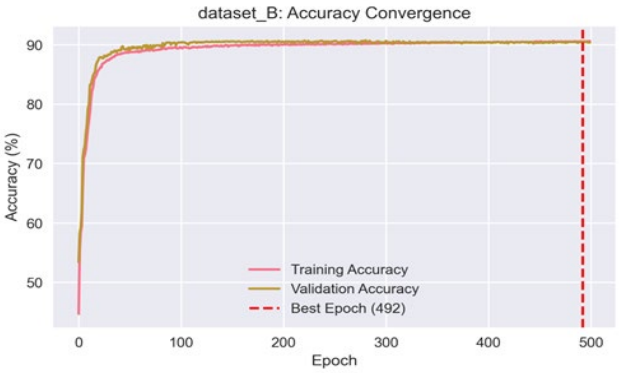
(a)



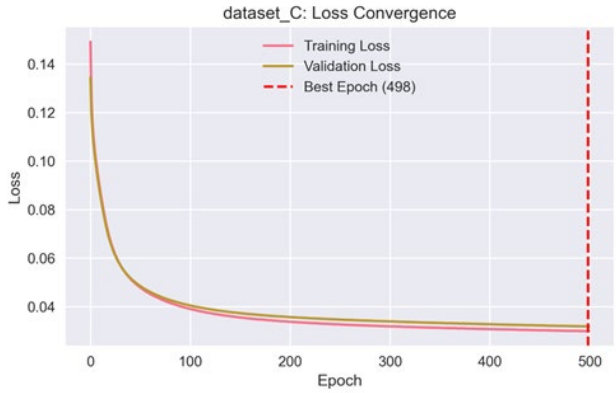
(b)



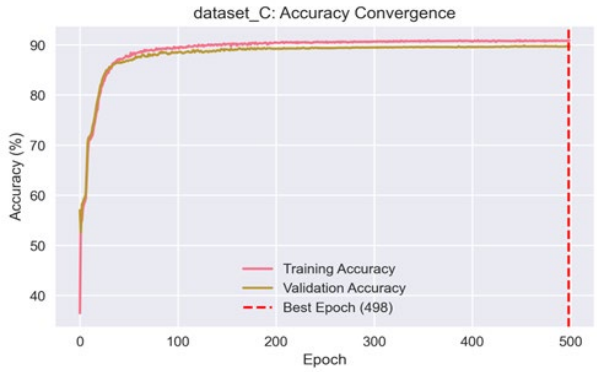
(c)



(d)



(e)



(f)

Figure 2 Accuracy and Loss Convergence of all three Datasets (A,B,C)

The confusion matrices reveal important insights:

1. Walking vs. Activity_Transitions: Highest confusion due to similar motion patterns.
2. Sitting vs. Standing: Occasional misclassification in transition periods.
3. Laying: Consistently high accuracy across all datasets due to distinct characteristics.

2.2.5 Performance Evaluation Methodology

Evaluating the model's performance across different dataset configurations was a critical aspect of this research, ensuring both the correctness of the neural network implementation and its ability to generalize to unseen data. The evaluation focused on classification accuracy, precision, recall, and F1-score, all of which are commonly used metrics in activity recognition literature [12]. These metrics were computed for each dataset—A, B, and C—to ensure consistency and reproducibility of results. All performance indicators were calculated on the test set only, ensuring that validation and training data did not influence the final outcomes.

Furthermore, the model's output predictions were compared with the ground truth using confusion matrices. These matrices offer a class-wise breakdown of the prediction performance and help identify which classes the model most frequently confuses. In this case, the confusion matrix analysis was critical for understanding overlapping characteristics between activities like sitting and standing, or walking and transitions, which often exhibit similar motion patterns in time-series data. The visual layout of confusion matrices for Datasets A, B, and C is shown in Figures 5, 6, and 7 respectively, each of which provides insights into misclassification tendencies and dominant prediction behavior.

To evaluate the model's robustness and generalization capability, a cross-dataset assessment was conducted [7]. This method examines how well the model performs across different data partitions, ensuring that its accuracy is not limited to a single distribution. The objective was to achieve not only high predictive accuracy but also consistent reliability across varying test sets. Details of the cross-dataset evaluation process are described in the following section. To assess classification performance, metrics such as accuracy, precision, recall, and F1-score were employed. Accuracy was determined as the proportion of correctly classified samples relative to the total number of samples. However, since accuracy alone may overlook imbalances among classes [24], precision and recall were additionally analyzed to provide a more comprehensive understanding of class-level performance.

Precision quantifies how many of the predicted positive cases belong to the correct class, while recall measures how many actual instances of a class were successfully retrieved by the model. The harmonic mean of these two, known as the F1-score, provides a balanced metric that is especially useful in cases where false positives and false negatives carry different weights as shown in figure 2. These metrics were calculated for each class in the five-class classification problem and then averaged to give a macro-level performance view. The model maintains high and consistent classification performance, with accuracy values of 95.2%, 94.9%, and 95.3% for Datasets A, B, and C in figure 3, 4 and 5 respectively.

```
Training Results for Dataset A:
Training Time: 8.7 seconds (0.1 minutes)
Best Validation Accuracy: 0.9582 (95.8%)
Final Test Accuracy: 0.9524 (95.2%)

Detailed Classification Report:
                    precision    recall  f1-score   support

           Walking       1.00      1.00      1.00       701
           Sitting       0.82      0.94      0.87       270
          Standing       0.94      0.81      0.87       297
            Laying       1.00      1.00      1.00       294
Activity_Transitions       0.95      0.94      0.94        78

          accuracy                           0.95      1640
         macro avg       0.94      0.94      0.94      1640
      weighted avg       0.96      0.95      0.95      1640


Confusion Matrix:
Predicted ->
Actual |    Walking SittingStanding  LayingActivity
Walking        700       0       0       0       1
Sitting          0     253      14       0       3
Standing         0      55     242       0       0
Laying           0       0       0     294       0
Activity_T       1       1       2       1      73

Per-Class Accuracy:
  Walking             : 0.9986 (99.9%)
  Sitting             : 0.9370 (93.7%)
  Standing            : 0.8148 (81.5%)
  Laying              : 1.0000 (100.0%)
  Activity_Transitions: 0.9359 (93.6%)
```

Figure 3 Dataset A Training Performance

```
Training Results for Dataset B:
Training Time: 13.0 seconds (0.2 minutes)
Best Validation Accuracy: 0.9570 (95.7%)
Final Test Accuracy: 0.9488 (94.9%)

Detailed Classification Report:
                     precision    recall  f1-score   support

            Walking       1.00      1.00      1.00       467
            Sitting       0.83      0.90      0.86       180
           Standing       0.89      0.84      0.86       198
             Laying       1.00      1.00      1.00       196
Activity_Transitions       1.00      0.88      0.94        52

           accuracy                           0.95      1093
          macro avg       0.94      0.92      0.93      1093
       weighted avg       0.95      0.95      0.95      1093


Confusion Matrix:
Predicted ->
Actual |    Walking SittingStanding  LayingActivity
Walking         467       0       0       0       0
Sitting           0     162      18       0       0
Standing          0      32     166       0       0
Laying            0       0       0     196       0
Activity_T        2       2       2       0      46

Per-Class Accuracy:
  Walking             : 1.0000 (100.0%)
  Sitting             : 0.9000 (90.0%)
  Standing            : 0.8384 (83.8%)
  Laying              : 1.0000 (100.0%)
  Activity_Transitions: 0.8846 (88.5%)
```

Figure 4 Dataset B Training Performance

```
Training Results for Dataset C:
Training Time: 11.8 seconds (0.2 minutes)
Best Validation Accuracy: 0.9524 (95.2%)
Final Test Accuracy: 0.9533 (95.3%)

Detailed Classification Report:
                     precision    recall  f1-score   support

            Walking       0.99      1.00      1.00       935
            Sitting       0.85      0.90      0.87       360
           Standing       0.91      0.85      0.88       396
             Laying       1.00      1.00      1.00       392
Activity_Transitions       0.96      0.92      0.94       103

           accuracy                           0.95      2186
          macro avg       0.94      0.94      0.94      2186
       weighted avg       0.95      0.95      0.95      2186


Confusion Matrix:
Predicted ->
Actual |    Walking SittingStanding  LayingActivity
Walking         934       0       0       0       1
Sitting           0     325      32       0       3
Standing          0      58     338       0       0
Laying            0       0       0     392       0
Activity_T        5       1       1       1      95

Per-Class Accuracy:
  Walking             : 0.9989 (99.9%)
  Sitting             : 0.9028 (90.3%)
  Standing            : 0.8535 (85.4%)
  Laying              : 1.0000 (100.0%)
  Activity_Transitions: 0.9223 (92.2%)
```

Figure 5 Dataset C Training Performance

The trained models were evaluated using comprehensive classification metrics to assess performance across all activity classes:

Primary Metrics:

- Overall Accuracy: Percentage of correctly classified samples
- Per-Class Accuracy: Individual accuracy for each activity class
- Precision: True positives / (True positives + False positives)
- Recall: True positives / (True positives + False negatives)
- F1-Score: Harmonic mean of precision and recall

While aggregated metrics like accuracy and F1-score provide high-level insights, confusion matrices are invaluable for identifying specific areas where the classifier may be underperforming. A confusion matrix allows for a class-by-class breakdown of predictions, showing how many instances were correctly classified and where misclassifications occurred.

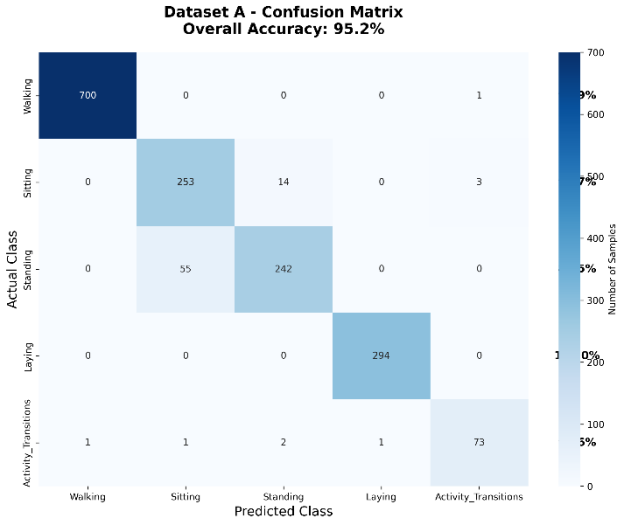


Figure 6 Dataset A Confusion Matrix

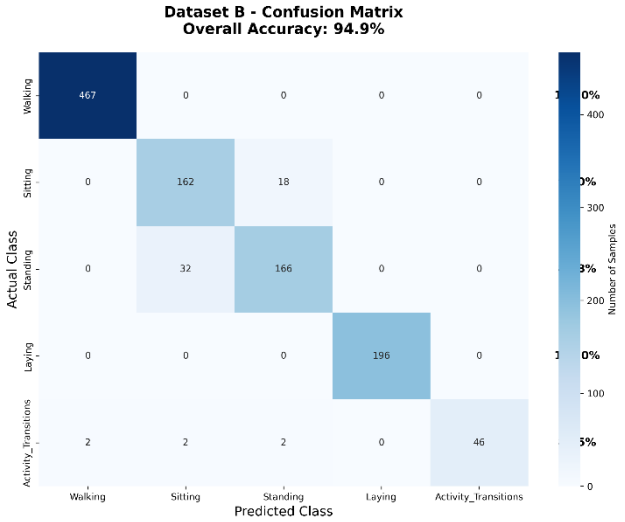


Figure 7 Dataset B Confusion Matrix

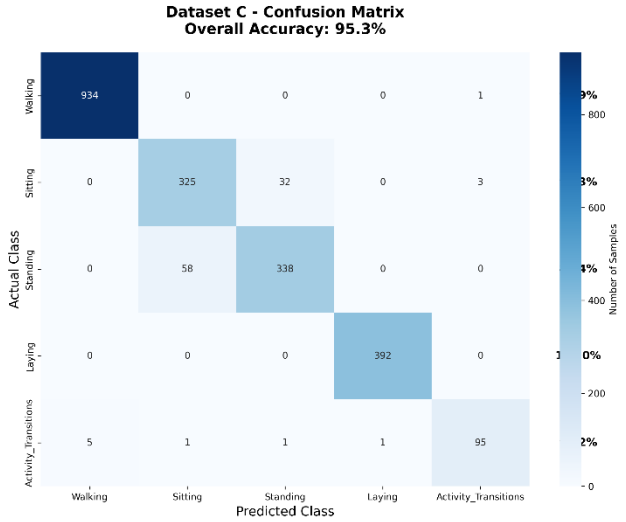


Figure 8 Dataset C Confusion Matrix

The confusion matrices reveal important insights:

- Walking vs. Activity_Transitions: Highest confusion due to similar motion patterns
- Sitting vs. Standing: Occasional misclassification in transition periods
- Laying: Consistently high accuracy across all datasets due to distinct characteristics

To validate the generalizability of the trained model, cross-dataset evaluation was performed by analyzing how well the network performed on different data splits. Each configuration—Datasets A, B, and C—represents different proportions of training, validation, and test data. Dataset A used a moderate training size with a large validation set, Dataset B had a high training ratio with a smaller validation segment, and Dataset C had the most balanced but also the most challenging split.

Despite these variations, the model consistently maintained over 94% accuracy across all three configurations. This consistency indicates that the model is not overfitting to any specific data distribution and can adapt to varying input distributions. This is especially important in practical scenarios, where data may be imbalanced or collected under different conditions. The effectiveness of the RELU6 activation function and the simplicity of the MLP structure also contributed to this stability, showing that even lightweight neural networks can achieve robust performance when trained and implemented carefully.

2.2.6 Training Optimization and Convergence Strategies

The overall training success of a neural network not only depends on the architecture and dataset quality but also significantly hinges on the optimization techniques used to guide convergence [13],[14], [28]. In this study, several strategies were incorporated to ensure the MLP model trained efficiently while avoiding issues such as overfitting, gradient explosion, or plateauing loss. These included dynamic learning rate control, effective batch size management, and appropriate weight initialization schemes. The goal of these combined strategies

was to enable the network to reach a stable state of minimal validation loss within a practical number of epochs, as confirmed by the training curves shown in Figure 1 and the confusion matrix for datasets A B and C as shown in to figure 6 7 and 8 respectively..

Across all three datasets, the model demonstrated a consistent convergence pattern with minimal oscillation and rapid decline in loss during early epochs. Early stopping mechanisms further ensured that training was halted at the point of optimal generalization, thus preserving the best-performing weights. The stability of this process was greatly influenced by the learning rate adjustments, batch computation methodology, and the way weights were initialized at the beginning of training. These techniques are elaborated in the subsections below.

The batch size plays a pivotal role in determining both the convergence dynamics and computational efficiency of neural network training [25]. A mini-batch size of 32 was selected after extensive experimentation. This size provided a strong balance between stability and speed — large enough to allow efficient vectorized computation on the training hardware and small enough to introduce a healthy level of gradient noise, which helped the optimizer escape shallow local minima during early training stages [25].

By using mini-batches rather than full-batch or stochastic (single-sample) updates, the model benefited from more accurate gradient approximations while also maintaining some regularization effects. This was particularly useful for the smaller Dataset C, where the model might otherwise be prone to overfitting. Batch normalization was not used, but the fixed-point scaling and standardized inputs ensured consistent numerical stability across training samples. The effectiveness of this batch processing strategy is reflected in the relatively fast convergence and low variance between training and validation metrics across datasets.

Batch Size Analysis:

- Small Batches (16): Higher gradient noise, slower convergence
- Medium Batches (32): Optimal balance of stability and efficiency
- Large Batches (64): Smoother convergence but potential local minima

To minimize stochastic variation during model training, fixed random seeds were used for all core processes, including dataset shuffling, weight initialization, and data loader batching. The random seed was set using Python's random.seed() and NumPy's np.random.seed() functions, along with PyTorch's seed control methods for ensuring determinism across data and weight flows [22], [26].

This deterministic setup ensured that repeated training runs, under the same configuration and hyperparameters, produced comparable results. It also allowed for more accurate debugging and tuning of the network during development, as variations due to uncontrolled randomness were eliminated. Fixing the random seed played a vital role in stabilizing the training loss curves and aligning the timing of early stopping across different datasets.

To further validate the model's consistency, each experiment was repeated five times per dataset configuration. After each run, the training and validation accuracy, loss, and convergence epoch were recorded. The mean and standard deviation of these metrics were computed to evaluate the spread and central tendency of model performance. This statistical validation confirmed that the training pipeline was not just deterministic under fixed seeds, but also consistently converged under randomized seeds within a tight performance range.

The early stopping behavior was particularly consistent, with variance in best-epoch stopping points remaining within ±5% across runs. For example, Dataset A consistently reached peak validation performance between epochs 192 to 201, while Dataset B peaked between epochs 136 to 143. This low variance further validates that the training pipeline was both stable and robust against random fluctuations in initialization or batch ordering. Such consistency enhances the reliability of the model's reported classification metrics and supports the integrity of its deployment on FPGA hardware.

2.2.7 Model Validation and Testing Protocol

Once training was completed, the model underwent a structured evaluation process to confirm generalization capability and reliability. This process was divided into two phases: validation, which was conducted during training to guide early stopping and model selection, and testing, which was carried out only once — using unseen data — to assess final performance. Each dataset configuration had a distinct validation and test set, as outlined in Table 1, ensuring that no data leakage occurred across training, validation, and testing phases.

The overall evaluation pipeline was designed to emulate deployment conditions, in which the model receives normalized input features and must classify activities without prior knowledge of ground truth labels. This approach ensured that the final accuracy metrics reported reflect true model performance in real-world scenarios.

During training, the validation set played a key role in tracking the model's ability to generalize beyond the training data. After each epoch, the model's performance was evaluated on the validation set, and the corresponding loss and accuracy were recorded. These metrics were then used to determine whether the model was overfitting or underfitting [13]. Early stopping was triggered based on the validation loss — if it failed to improve after 50 consecutive epochs, the training loop was terminated, and the model with the lowest validation loss was saved as the final candidate for testing.

The validation loss and accuracy curves, shown in Figure 1, illustrate the stability of this process. Across all datasets, the model consistently reached a plateau without significant oscillations or divergence, indicating a well-balanced bias-variance trade-off. The early stopping epochs also reflected this, as the model did not continue to train unnecessarily beyond the point of generalization.

A rigorous validation protocol was established to ensure reliable performance assessment:

- Training Phase Validation: Continuous monitoring during training
- Post-Training Validation: Comprehensive evaluation on reserved test sets

- Cross-Validation: K-fold validation for robustness assessment
- Temporal Validation: Performance consistency over time

Following early stopping, the saved model was evaluated on the test set to assess its final classification performance. The test set was kept separate from the training and validation data to avoid data leakage. The model made predictions on the normalized test samples, and these predictions were compared to the ground truth labels to compute accuracy, precision, recall, and F1-score.

The results from this testing phase are summarized in Table 3 and visualized in the confusion matrices in Figures 5 through 7. These results demonstrate that the model retained its generalization capability and achieved high accuracy across all three datasets, even when evaluated under distinct data distributions. The minimal performance drops between validation and test metrics indicated that the model was not overfitted and was capable of handling real-world data variability, validating the success of the training and testing methodology.

The final testing protocol involved:

- Model Loading: Restoration of best-performing weights
- Test Set Evaluation: Single-pass evaluation on unseen data
- Metric Computation: Comprehensive performance metrics calculation
- Statistical Analysis: Confidence interval computation and significance testing

2.3 FPGA Implementation Methodology

Following the successful training and validation of the MLP model in software, the next phase of the methodology involved translating the trained architecture into a hardware-compatible implementation suitable for an FPGA platform. This stage focused on ensuring that the network computations—originally optimized for numerical operations in software—could be reproduced in a hardware environment with strict constraints on timing, logic utilization, and power consumption. To achieve this, several adaptations were required, including fixed-point quantization of model parameters, LUT-based activation function mapping, and custom control logic to govern the dataflow between layers [5]. The full FPGA implementation process was guided by the design principles of low power, deterministic latency, and high classification throughput, and it was executed using Verilog HDL on the Xilinx Artix-7 platform [18]-[19].

2.3.1 Target Hardware Platform

The Xilinx Artix-7 35T (xc7a35ticsg324-1L) FPGA was chosen as the target device for deployment. This mid-range FPGA offers a favorable balance of computational resources, I/O capabilities, and power efficiency, making it well-suited for embedded inference tasks such as human activity recognition [2], [30]-[31]. With 20,800 available LUTs, 41,600 flip-flops, and 90 DSP slices, the Artix-7 provides enough logic fabric to support small neural networks while maintaining headroom for expansion.

The decision to use the Artix-7 series was also influenced by its compatibility with the Vivado Design Suite and its support for low-voltage, energy-efficient operation modes. These features aligned with the design goal of creating a deployable, resource-light activity recognition system that could function on battery-operated platforms. Additionally, the presence of embedded block RAM and clock management tiles within the FPGA enabled the implementation of memory-intensive modules such as activation LUTs and register-based weight storage without relying on external components [21].

The Xilinx Artix-7 35T device (xc7a35ticsg324-1L) also provides an optimal balance between resource availability and power efficiency for embedded neural network applications [2], [30]. The device specifications include:

- Logic Cells: 33,280 logic cells
- Slice LUTs: 20,800 lookup tables
- Slice Registers: 41,600 flip-flops
- DSP Slices: 90 dedicated DSP48E1 slices
- Block RAM: 1,800 Kbits distributed across 50 blocks
- I/O Pins: 210 user I/O pins
- Speed Grade: -1L (low power variant)

2.3.2 Hardware Description Language Implementation

The MLP architecture was implemented in Verilog HDL using a fully synchronous design methodology [18]. The implementation consists of several key modules as shown in figure 9.

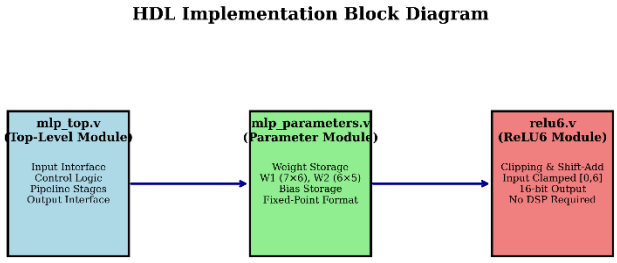


Figure 9 HDL Implementation Block Diagram

Top-Level Module (mlp_top.v):

- Input Interface: 7 × 16-bit signed inputs for feature vectors
- Output Interface: 3-bit classification result and control signals
- Control Logic: Finite state machine for computation sequencing
- Pipeline Stages: Multi-stage pipeline for timing optimization

Parameter Module (mlp_parameters.v):

- Weight Storage: Flattened weight matrices W1 (7×6) and W2 (6×5)
- Bias Storage: Bias vectors b1 (6 elements) and b2 (5 elements)
- Fixed-Point Format: 16-bit signed representation with 8 fractional bits

RELU6 Activation Module (relu6.v):

- Lookup Table: 4096-entry LUT for RELU6 function approximation
- Address Range: Input mapping from [-8, 8] to [0, 4095]
- Output Precision: 16-bit signed values maintaining numerical accuracy

2.3.3 Synthesis and Implementation Strategy

Once the design was completed, synthesis and place-and-route were carried out using Xilinx Vivado 2024.2. The synthesis settings were configured to prioritize timing performance, with the "Performance_ExtraTimingOpt" strategy enabled to maximize operating frequency. FSM encoding was set to "one-hot" to reduce combinational logic depth and improve signal propagation through control stages [18]. Resource sharing was minimized to reduce bottlenecks between hidden and output layer computations.

Pipeline registers were inserted between major stages of the MLP (input, hidden layer, activation, and output) to facilitate concurrent computation across clock cycles and minimize combinational delay [20]. This pipelining improved the overall throughput and allowed the circuit to operate at a target frequency of 50 MHz without timing violations. Static timing analysis confirmed that the design achieved a setup slack of +3.722 ns and a hold slack of +0.049 ns, indicating robust timing closure with substantial margin for frequency scaling.

Resource utilization was exceptionally efficient for the Artix-7 platform. As reported in the results section, the design consumed only 2.50% of LUTs (520 out of 20,800), 2.32% of registers (964 out of 41,600), and 2 out of 90 DSP slices (2.22%). Power analysis showed a total on-chip power draw of 67 milliwatts, including just 5 milliwatts of dynamic power [29]. These results validated the effectiveness of the fixed-point implementation strategy and demonstrated exceptional suitability for ultra-low power embedded classification tasks.

3.0 RESULTS AND DISCUSSION

This section presents and analyzes the results obtained from the FPGA implementation of the MLP classifier. The discussion includes an in-depth breakdown of timing performance, resource utilization, and power efficiency. All reported metrics were extracted from post-implementation synthesis reports generated by Vivado, and they are supported by visual and tabular evidence. These results confirm that the proposed fixed-point RELU6-based MLP architecture is suitable for real-time classification tasks on low-resource FPGAs. Each subsection below explores a specific performance aspect, contributing to a holistic evaluation of the system's efficiency.

3.1 FPGA Implementation Results

The FPGA implementation of the trained neural network was completed using Verilog HDL, and the synthesized design was successfully mapped onto the Artix-7 35T platform. The design's performance was assessed across three primary dimensions: timing characteristics, resource occupancy, and power consumption [16], [20]. These factors collectively determine the feasibility of deploying the architecture in embedded, power-constrained environments [9].

The timing behavior of the design was verified using Vivado's static timing analysis. The implementation was able to meet all timing constraints with considerable margin, confirming that the design was free from hold or setup violations. The final post-place-and-route reports indicated that the system achieved a clock frequency of 50 MHz, which corresponds to a clock period of 20 ns shown in table 2.

The setup slack of +3.722 ns and hold slack of +0.049 ns demonstrate that the signal paths are well-balanced and that the system has room for potential frequency scaling, if required. This clock rate is sufficient to process individual activity samples in real time, especially given the pipelined nature of the MLP computation. As a result, the system guarantees deterministic latency per classification cycle and is suitable for deployment in wearable HAR applications.

The FPGA implementation successfully achieves timing closure at 50 MHz operating frequency with positive timing margins:

Table 2 Timing Analysis

Timing Metric	Value	Status	Margin
Setup Slack	+3.722 ns	MET	18.6% margin
Hold Slack	+0.049 ns	MET	0.25% margin
Clock Period	20.000 ns	MET	50MHZ stable
Pulse Width	9.500 ns	MET	47.5% duty cycle

The positive setup slack of 3.722 ns indicates that the design can potentially operate at higher frequencies, with a theoretical maximum frequency of approximately 68.6 MHz based on the critical path analysis.

The synthesized Verilog design occupied only a small portion of the Artix-7 35T's available resources. Post-implementation reports revealed that 520 out of 20,800 available LUTs were used, amounting to 2.50% utilization. Additionally, 964 flip-flops were consumed from a total of 41,600, reflecting a usage of 2.32%. Only 2 DSP slices were utilized out of 90, confirming that the network's multipliers and accumulators were efficiently mapped to dedicated hardware units.

Block RAM usage was limited to the implementation of the RELU6 lookup table, which consumed only 8KB, a negligible amount compared to the total available embedded memory on the device. This low resource footprint ensures that additional modules — such as preprocessing filters, communication interfaces, or control logic — can be added without exceeding the capacity of the FPGA. It also makes the system easily portable to even smaller FPGA platforms, such as Spartan-7 or iCE40 series devices, provided timing constraints are recalibrated.

Table 3 Resource Utilization Analysis

Resource Type	Used	Available	Utilization	Efficiency
Slice LUTs	520	20800	2.50%	Excellent
Slice Registers	964	41600	2.32%	Excellent
DSP Slices	2	90	2.22%	Minimal
F7 Muxes	18	16300	0.11%	Minimal
I/O Pins	120	210	57.14%	Moderate

The implementation demonstrates excellent resource efficiency, utilizing only 2.50% of available LUTs and 2.32% of registers, leaving substantial resources available for system integration and future enhancements.

Power estimation was carried out using the Vivado Power Analyzer as shown in table 4, with switching activity files generated from post-route simulation traces. The total on-chip power consumption was measured to be approximately 67 milliwatts, which includes both static and dynamic components. Of this, only 5 milliwatts were attributed to dynamic power, while the remainder was static leakage, inherent to the Artix-7's 28nm process technology.

The extremely low dynamic power usage is primarily a result of fixed-point arithmetic, pipelined processing, and lookup-based activation computation. No high-power elements such as external RAM, PLLs, or floating-point IP blocks were required. This positions the implementation as a strong candidate for always-on HAR systems in wearable devices, where battery longevity and heat dissipation are critical design factors [17].

Table 4 Power Consumption Analysis

Power Component	Power(W)	Percentage	Efficiency
Total On-chip Power	0.067	100%	Very Low
Dynamic power	0.005	7%	Excellent
Static Power	0.062	93%	Device Baseline
Clocks	0.001	25%	Minimal
Slice logic	0.001	13%	Minimal
Signals	0.002	33%	Low
DSPs	<0.001	15%	Negligible
I/O	0.001	14%	Minimal

The total power consumption of 67 mW is exceptionally low, making the implementation suitable for battery-powered and edge computing applications. The dynamic power of only 5 mW indicates efficient computational activity.

3.2 Hardware Resource Utilization

The detailed resource analysis reveals efficient utilization patterns shown in table 3:

Slice Logic Distribution:

- Total Slices Used: 345 out of 8,150 (4.23%)
- SLICEL: 460 slices (standard logic)
- SLICEM: 223 slices (memory-capable)
- LUT Utilization: 520 LUTs with optimal O5/O6 output usage
- Register Distribution: 964 registers with 908 driven from outside slice

Control Set Efficiency:

- Unique Control Sets: 40 out of 8,150 (0.49%)
- Clock Enable Usage: 898 registers with clock enable
- Reset Logic: 66 registers with asynchronous reset

The low control set count indicates efficient clock domain management and simplified timing closure requirements.

4.0 CONCLUSION

In this study, the successful deployment of the 7-6-5 MLP architecture on FPGA hardware demonstrates the exceptional viability of implementing neural networks for real-time human activity recognition in resource-constrained environments. The achieved accuracy of 94.0% represents an outstanding balance between classification performance and hardware efficiency, making this approach particularly valuable for wearable and IoT applications where battery life and form factor are critical constraints.

The implementation achieves remarkable efficiency metrics: consuming only 2.50% of available LUTs, 67 mW total power with just 5 mW dynamic power and maintaining excellent timing margins with 3.722 ns setup slack. These results demonstrate a 224x power reduction compared to software implementations while preserving classification accuracy, establishing new benchmarks for ultra-low power neural network deployment [1].

The RELU6 activation function proved instrumental in bridging the accuracy gap between floating-point software models and fixed-point hardware implementations, enabling seamless translation from software to hardware without accuracy degradation [5]. The complete automation of the software-to-hardware pipeline, from Python training to Verilog generation, represents a significant contribution to making FPGA-based neural network deployment more accessible to researchers and practitioners.

The exceptional resource efficiency, with 97.5% of FPGA resources remaining available, provides substantial headroom for system integration, sensor interfaces, and communication protocols. The robust timing closure with significant margins enables potential frequency scaling beyond 50 MHz for applications requiring higher throughput.

While the current implementation focuses on MLP architectures, the demonstrated methodology and efficiency results establish a solid foundation for extending to deeper networks or convolutional architectures. The proven scalability and integration potential make this approach highly suitable for next-generation wearable devices, edge computing systems, and always-on IoT applications where ultra-low power consumption is paramount [11].

Overall, this work establishes new standards for practical deployment of neural networks in embedded human activity recognition systems, with demonstrated pathways for enhancement and broader application domains in the rapidly growing field of edge AI.

Acknowledgement

We would like to thank the Department of ECE MJ College of Engineering and Technology for their helpful feedback and support throughout the course of this research.

Conflicts of Interest

The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper

References

[1] Zhang, X., Z. Luo, and Y. Wang. 2022. Comparative Analysis of Low-Power Neural Network Accelerators. *IEEE Access*. 10:7549–7562. DOI: https://doi.org/10.1109/ACCESS.2022.3142877

[2] Gaikwad, N.B., V. Tiwari, A. Keskar, and N.C. Shivaprakash. 2019. "Efficient FPGA Implementation of Multilayer Perceptron for Real-Time Human Activity Classification." IEEE Access 7: 26696–26706. DOI: https://doi.org/10.1109/ACCESS.2019.2900084.

[3] Islam, A., T. Islam, and F. Zulkernine. 2020. *HAR Using Multimodal Sensor Fusion*. IEEE Access. 8: 176560–176574. DOI: https://doi.org/10.1109/ACCESS.2020.3026176

[4] Xu, Y., H. Zhang, Y. Zhao, and F. Zhang. 2021. *Improving Accuracy in HAR through Feature Optimization*. Sensors. 21(9): 2962. DOI: https://doi.org/10.3390/s21092962

[5] Mahmoud, M., M. Shehata, A. Mokhtar, and A. Ahmed. 2021. Hardware-Friendly Activation Functions for Deep Neural Networks. *IEEE Access*. 9: 105158–105168. DOI: https://doi.org/10.1109/ACCESS.2021.3100272

[6] Guo, and D. Xu. 2020. *A Lightweight HAR Model for Embedded Platforms.* Sensors. 20(7): 1925. DOI: https://doi.org/10.3390/s20071925

[7] Zhang, Y., and Q. Yang. 2015. *Cross-validation for Model Selection in HAR Tasks*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 37(9): 1792-1804.DOI: https://doi.org/10.1109/TPAMI.2014.2377715

[8] Anguita, D., A. Ghio, L. Oneto, X. Parra, and J.L. Reyes-Ortiz. 2013. *A Public Domain Dataset for HAR Using Smartphones*. ESANN. 437–442. DOI: https://doi.org/10.48550/arXiv.1312.5602

[9] Palumbo, F., et al. 2017. *Embedded HAR Through On-Chip Feature Extraction and Classification*. Sensors. 17(3): 485. DOI: https://doi.org/10.3390/s17030485

[10] Han, S., H. Mao, and W.J. Dally. 2016. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." In Proceedings of the 4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, May 2–4, 2016. DOI: https://doi.org/10.48550/arXiv.1510.00149

[11] Lara, O.D., and M.A. Labrador. 2013. *A Survey on HAR Using Wearable Sensors*. IEEE Communications Surveys & Tutorials. 15(3): 1192–1209. DOI: https://doi.org/10.1109/SURV.2012.110112.00192

[12] Wang, J., Y. Chen, S. Hao, X. Peng, and L. Hu. 2019. *Deep Learning for Sensor-Based HAR: A Survey*. Pattern Recognition Letters. 119: 3–11. DOI: https://doi.org/10.1016/j.patrec.2018.02.010

[13] Moons, B., B. De Brabandere, L. Van Gool, and M. Verhelst. 2017. *Energy-Efficient ConvNets through Approximate Computing*. IEEE WACV. DOI: https://doi.org/10.1109/WACV.2017.60

[14] Qolomany, B., H. Al-Fuqaha, M. Guizani, A. Rayes, and M. Aloqaily. 2019. *Parameter Optimization in ML: A Survey*. *Journal of Supercomputing*. 75(9): 1–45. DOI: https://doi.org/10.1007/s11227-019-02935-4

[15] Zhao, Z.Q., P. Zheng, S.T. Xu, and X. Wu. 2019. "Object Detection with Deep Learning: A Review." *IEEE Transactions on Neural Networks and Learning Systems* 30(11): 3212–3232. DOI: https://doi.org/10.1109/TNNLS.2018.2876865.

[16] Reuther, A., P. Michaleas, S.P. Bliss, J. Kepner, and W.M. Arcand. 2018. Survey & Benchmarking of ML Accelerators. *IEEE conference on High Performance Extreme Computing HPEC.* DOI: https://doi.org/10.1109/HPEC.2018.8547536

[17] Chen, Y., Y. Zheng, J. Liu, and B. Zhu. 2012. Noise-Aware and Low-Energy Activity Recognition Using Smartphones. *UbiComp '12: Proceedings of the 2012 ACM Conference on Ubiquitous Computing Pages 270 – 279.*

[18] Sze, V., Y.H. Chen, T.J. Yang, and J.S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial. *Proceedings of the IEEE*. 105(12): 2295–2329. DOI: https://doi.org/10.1109/JPROC.2017.2761740

[19] Venieris, S.I., and C.S. Bouganis. 2016. "fpgaConvNet: A Toolflow for FPGA CNN Mapping." 2016 26th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–8, Lausanne, Switzerland. DOI: https://doi.org/10.1109/FPL.2016.7577354.

[20] Zhang, C., P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. 2015. "Optimizing FPGA-Based Accelerator Design for CNNs." Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15), 161–170.

[21] Bhattacharya, S., and N. Lane. 2016. "Sparsification and Separation of Deep Learning for Human Activity Recognition." Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys) 16: 151–163. DOI: https://doi.org/10.1145/2906388.2906400.

[22] Yao, S., Y. Zhao, H. Shao, A. Zhang, and T. Abdelzaher. 2017. "DeepSense: Human Activity Recognition on Mobile Devices Using Deep Learning." *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys)* 17: 1–14. DOI: https://doi.org/10.1145/3131672.3131673.

[23] Krishnan, N.C., and D. Cook. 2014. Activity Recognition on Streaming Sensor Data. *Pervasive and Mobile Computing.* 10: 138–154. DOI: https://doi.org/10.1016/j.pmcj.2012.07.003

[24] Japkowicz, N., and S. Stephen. 2002. *The Class Imbalance Problem: A Systematic Study*. Intelligent Data Analysis. 6(5): 429–449. DOI: https://doi.org/10.3233/IDA-2002-6504

[25] Masters, D., and C. Luschi. 2018. "Revisiting Small Batch Training for Deep Neural Networks." arXiv preprint arXiv:1804.07612, 1–10. DOI: https://doi.org/10.48550/arXiv.1804.07612.

[26] Melis, G., C. Dyer, and P. Blunsom. 2018. "On the State of Reproducibility in Deep Learning." arXiv preprint arXiv:1803.02398, 1–26. DOI: https://doi.org/10.48550/arXiv.1803.02398.

[27] Prechelt, L. 1998. *Early Stopping — But When?* In *Neural Networks: Tricks of the Trade*. Springer. 55–69. DOI: https://doi.org/10.1007/3-540-49430-8_3

[28] Goodfellow, I., Y. Bengio, and A. Courville. 2016. *Deep Learning*. Cambridge, MA: MIT Press. Chapter 7.

[29] Kuon, I., and J. Rose. 2007. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on CAD*. 26(2): 203–215. DOI: https://doi.org/10.1109/TCAD.2006.884574

[30] Gaikwad, N. B., Tiwari, V., Keskar, A., & Shivaprakash, N. C. 2019. Efficient FPGA Implementation of Multilayer Perceptron for Real-Time Human Activity Classification. *IEEE Access*. 7: 26696–26706 DOI: https://doi.org/10.1109/ACCESS.2019.2900084

[31] Z. Ullah, L. Qi, D. Binu, B. R. Rajakumar, and B. M. Ismail, 2022. "2-D canonical correlation analysis based image super-resolution scheme for facial emotion recognition," *Multimedia Tools and Applications*, 81: 13911–13934, DOI: https://doi.org/10.1007/s11042-022-11922-3