Engineering Applications of Artificial Intelligence 182 (2026) 116034

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

# Microcontroller neural architecture search: An automated framework for developing a fall detection system

Seyed Mojtaba Mohasel [a], John Sheppard [b], Lindsey K. Molina [c], Richard R. Neptune [c], Shane R. Wurdeman [d], Corey A. Pew [a,*]

[a] Montana State University, Mechanical and Industrial Engineering, Bozeman, MT, USA
[b] Montana State University, Gianforte School of Computing, Bozeman, MT, USA
[c] Walker Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX, USA
[d] Department of Clinical and Scientific Affairs, Hanger Clinic, Austin, TX, USA

ARTICLE INFO

*Keywords:*
Automated machine learning
Tiny machine learning
Neural architecture search
Pruning
Class imbalance
Fall detection
Lower limb amputee
Inertial measurement unit (IMU)

ABSTRACT

This article introduces microcontroller neural architecture search (MicroNAS), an automated tool specifically designed to create models optimized for microcontrollers with small memory resources. The Espressif Systems 32-bit SoC (ESP32) microcontroller, with 320 KB of memory, is used as the target platform. The artificial intelligence contribution lies in a novel method for optimizing one-dimensional convolutional neural network (1D CNN) and gated recurrent unit (GRU) architectures by considering the memory size of the target microcontroller as a guide. A comparison is made between memory-driven model optimization and traditional two-stage methods, which use pruning, to show the effectiveness of the proposed framework. To demonstrate the engineering application of MicroNAS, a fall detection system (FDS) for lower-limb amputees is developed as a pilot study. A critical challenge in fall detection studies, class imbalance in the dataset, is addressed. The results show that MicroNAS-optimized models achieved superior performance, with the GRU and 1D CNN achieving Fall-class F1-scores of $0.66 \pm 0.18$ and $0.64 \pm 0.17$ respectively, significantly outperforming ensemble methods ($0.44 \pm 0.13$) and H2O Automated Machine Learning ($0.22 \pm 0.07$). These improvements, made possible by the customized automated framework, represent a significant step forward in real-time FDS development. Biomechanists using body-worn sensors for activity detection can adopt the open-source code to design machine learning models tailored for microcontroller platforms with limited memory.

## 1. Introduction

Falls present a major health risk for individuals with lower limb amputation (Miller et al., 2001; Liu et al., 2017). Specifically, more than half of lower limb amputees report falling in the previous 12 months. Furthermore, of those reporting a fall, approximately 75% report multiple falls (Miller et al., 2001). Falls have the potential for multiple negative sequelae, including fractures, traumatic brain injuries, lacerations, sprains, hematomas, and even death (Terroso et al., 2014). More commonly, a fall may only result in minor injuries or bruises but can impact the person's confidence in their balance and mobility (Terroso et al., 2014). Consequently, they may limit their physical activity and social participation, leading to a decline in overall physical and emotional health.

Falls also pose a barrier to successful rehabilitation, whether it be physical or emotional injury. The extent to which falls delay or prevent successful rehabilitation of individuals with lower limb amputations is unknown. Currently, evaluations of falling frequency are primarily conducted via survey and self-report. However, the accuracy of patient recall over an extended period is questionable (Ganz et al., 2005; Cummings et al., 1988). A more quantitative method is critically needed to evaluate the number of falls experienced in the amputee community to objectively evaluate the magnitude of the problem and the effects of specific interventions on fall risk.[1] Multiple studies have developed simple, low-cost sensors that can be attached to an individual in order to detect fall events. The primary methodology for detecting falls is with the use of inertial measurement units (IMUs) that include some form of multi-axis accelerometers, gyroscopes and/or magnetometers. Wearable

* Corresponding author.
*E-mail address:* Corey.Pew@montana.edu (C.A. Pew).
[1] https://github.com/MojtabaMohasel/MicroNAS.

https://doi.org/10.1016/j.engappai.2026.116034
Received 7 April 2025; Received in revised form 28 April 2026; Accepted 14 August 2026

devices are uniquely suited for the amputee community as IMUs are often already incorporated into systems such as microprocessor knees and ankles (Dauriac et al., 2019). Data from sensors are monitored to look for motion events that are classified as falls in comparison to regular activity. Simple fall detection algorithms utilize metrics such as threshold detection and orientation change to distinguish falls from everyday activity (Chen et al., 2006; Bourke et al., 2007, 2008), however these often produce lower accuracy than required for real-world implementation. More complex algorithms are needed to accurately detect fall events with low false positive rates to provide clinically meaningful feedback for rehabilitation of individuals with lower-limb amputation.

Machine learning (ML) has become a vital tool across various engineering disciplines (Sakhaee et al., 2025; Pandiyan et al., 2025). In biomechanics, it is commonly applied to identify falls from raw sensor data (Rastogi et al., 2021) and classify activities of daily living (ADL) using body-worn sensors (Ramanujam et al., 2021a). In addition, integrated tools in MATLAB, Python, and other software have made it easy to create ML models with any dataset. However, not all biomechanists are trained in appropriate ML techniques and can easily neglect best practices when developing ML models (Halilaj et al., 2018).

Here we identify three main principles often overlooked when developing fall detection classifiers:

### 1.1. Data division

Leave-one-participant-out is often the preferred approach in medical settings. To ensure real-world applicability, data from different participants should be individually divided into training, validation, and test sets (Halilaj et al., 2018). The training set determines model parameters (weights for neural networks and splitting rules for decision trees). The validation set is used to monitor the model's performance on data it has not seen during training. The validation set is used to identify model hyperparameters (filter size, number of filters, and learning rate for a neural network and maximum depth for a decision tree). The test set assesses the model's performance and estimates its real-world capabilities and should only include data from an individual participant that is not a part of the training and validation sets (entirely unseen). Comparisons between different ML methods must utilize consistent sets (train, validation, test) and apply appropriate statistical comparisons (Rainio et al., 2024).

### 1.2. Class imbalance

Utilizing ML models that assume equal sample distribution across classes is inappropriate as the number of ADL samples significantly outnumbers fall samples leading to poor classification performance for the minority class. This issue is commonly referred to as class imbalance, and it is quantified using the imbalance ratio (IR), the number of majority samples divided by the number of minority samples (Chen et al., 2021). Models specifically designed to handle class imbalance or methods capable of addressing class imbalance must be used (Mohasel et al., 2026a, Mohasel et al., 2026b).

### 1.3. Application

In resource-constrained environments (e.g., when using microcontrollers), researchers encounter hardware limitations for runtime memory capacity (Rastogi et al., 2021; Mohasel et al., 2026a). Attaining high sensitivity and specificity for fall detection without addressing the capabilities of microcontrollers that will run the ML model is insufficient. A critical consideration is the model run-time size, as microcontrollers have limited memory. Researchers should develop models that account for this constraint or utilize techniques like compression (Risso et al., 2022), quantization (Hubara et al., 2018), and pruning (Zhu et al., 2017) to reduce the model size and then report the appropriate performance metrics.

Inattention to these principles may result in misuse of ML techniques. Models may demonstrate excellent performance on tested data from a laboratory setting but may not be feasible for deployment and generalize poorly to real-world environments. While biomechanists can work with ML experts to navigate best practices, the interactive process can be time-consuming and require significant effort from both parties (Karmaker Santu et al., 2020). In addition, since each dataset has unique characteristics, the development of diverse methods, including neural networks and Ensemble models (Zebiah et al., 2023), is essential to find the best fall detection model.

Automated machine learning (AutoML) can address these challenges by automating the entire ML pipeline (He et al., 2021; Mohasel et al., 2026c). AutoML is a framework designed to automate the application of ML to real-world problems (Karmaker Santu et al., 2020). AutoML frameworks streamline various stages of the ML pipeline development, including data preprocessing, model selection, hyperparameter tuning, and performance evaluation. Introducing an AutoML framework, particularly in the context of biomechanical systems like fall detection, would represent a significant step forward in making advanced technology accessible to non-experts. With automated tools, biomechanists can focus more on the experimental design, data collection, results analysis, and clinical applications of fall detection, while spending less time on the complexities of model development.

AutoML leverages currently available, advanced algorithms to empower biomechanists to build ML applications appropriately, without requiring extensive statistical or ML knowledge (Zöller et al., 2021). However, AutoML tools have limitations: (a) H2O AutoML, Auto-Sklearn, and AutoGluon do not handle multivariate IMU time-series data; (b) Akkio and Azure may support IMU data, but they require a subscription; and (c) Mcfly (Van Kuppevelt et al., 2020) handles time-series data but does not offer models optimized for resource-constrained environments. Therefore, current AutoML tools are not well-suited to address the unique requirements of developing an FDS.

Deep Learning (DL) models are the leading approaches for an FDS (Islam et al., 2020); however, automating the entire pipeline of a DL model for a microcontroller is challenging. For example, automating a DL pipeline requires expertise in signal processing, neural networks, and optimization (Chitty-Venkata et al., 2022). In addition, DL models involve complex algebraic operations, leading to high power consumption and long response times when executed on wearable devices with limited resources (Salah et al., 2022). These challenges have sparked the evolution of modern DL model training called micro controller unit-aware neural architecture search (MCU-aware NAS). MCU-aware NAS, a subfield of AutoML, has been developed to create model architectures compatible with small memory footprints of microcontrollers (Lin et al., 2020a). Despite the initial advances of MCU-aware NAS, it is still in its infancy, and model architecture design specifically tailored for microcontrollers is scarce (Lin et al., 2020a). The present work seeks to address several knowledge gaps in this domain.

Knowledge Gap 1: An open-source AutoML pipeline that handles time series data, class imbalance, and memory constraints in a microcontroller-based FDS is needed.

One AutoML approach for creating small memory footprint models is Tiny-NAS (Lin et al., 2020a). The principle behind TinyNAS is that higher Floating-Point OPeration per Second (FLOPS) architectures produce models with higher accuracy. FLOPS represent the number of additions, subtractions, multiplications, and divisions that a computer can perform in 1 s. TinyNAS employs two distinct stages to identify the best model within a given space. In stage 1, the design space (Radosavovic et al., 2019, 2020) is analyzed by sampling architectures, resulting in the selection of a part of the design space that accommodates higher FLOPS while meeting the memory requirements (Lin et al., 2020a). In stage 2, TinyNAS selects the best model by optimizing the memory size through pruning (Lin et al., 2020a). Pruning results in a smaller model

size facilitating faster inference speed, lower memory requirements, and improved efficiency on resource-constrained devices. However, there is a tradeoff between model pruning and F1-score (Tran et al., 2022). Improved NAS designs are essential to enable the deployment of KB-sized neural networks without sacrificing F-score on resource-limited microcontrollers (Ray, 2022). Searching architectures by expanding the model first and pruning it afterward in two stages can reduce the F1-score performance depending on the pruning rate (Tran et al., 2022; Liebenwein et al., 2021). Therefore, considering one comprehensive stage for model creation from the beginning is potentially a more efficient search strategy.

Knowledge Gap 2: A more direct and comprehensive, single-stage approach is currently not available to optimize architectures for specific microcontroller deployment without pruning.

Traditionally, studies have developed FDS for the elderly or populations without physical impairment (Wang et al., 2020). Few studies have focused on developing FDS for individuals with lower limb amputations. In one study, control participants without physical disability were used as substitutes for lower limb amputees to develop ML models for an FDS (Shawen et al., 2017). The ML model results indicated that there is no statistical difference between a model trained on a control population and tested on a lower limb amputee population versus a model trained and tested on amputees. Nevertheless, the use of control participants as a proxy for lower limb amputees in developing ML models for FDS remains contentious. The study's results were obtained with sensor placements on the waist, in a pocket, and in the hand. However, a sensor located on the prosthetic leg for amputees might be sensitive to the training data population (non-amputee or amputee). It is unclear if an ML model developed on control participants will perform as well on amputees as it does on controls.

Knowledge Gap 3: A statistical experiment is needed to validate the use of control participants for developing an ML model for an FDS for the amputee community.

The primary objective of this work was to develop an automated ML framework called MicroNAS. Our secondary objective was to develop a fall detection system for individuals with lower limb amputation. We developed MicroNAS and tested it by developing a fall detection algorithm as a test case. MicroNAS is a custom neural architecture search tool that efficiently explores feasible space and creates deployable models using the microcontroller's memory size as a guide. In this work, we focused on utilizing the Espressif Systems 32-bit SoC (ESP32) (Babiuch et al., 2019), a microcontroller known for its cost-effectiveness, low power consumption with compact board size, facilitating its incorporation into a lower limb prosthesis. Our emphasis was on ESP32-S2 series with 320 KB runtime memory (SRAM). The body-worn FDS in this work is specifically designed for the amputee community and utilizes a single sensor placed on the pylon of a lower-limb prosthesis. The FDS leverages AutoML techniques for neural network model creation, alleviating the burden on researchers to develop models manually. Ultimately, the MicroNAS tool can identify a suitable architecture for neural network deployment on low-cost microcontrollers such as the ESP32 (Babiuch et al., 2019).

To evaluate MicroNAS's ability to address the knowledge gaps, we introduce the following three hypotheses.

**Hypothesis 1.** Neural network models optimized with MicroNAS achieve higher performance, based on F1-score, compared to ensemble models or open-source AutoML tools capable of handling class imbalance automatically and operating with minimal memory requirements for ESP32.

**Hypothesis 2.** Neural network model architectures generated by MicroNAS, without pruning, will achieve higher F1-scores and lower memory sizes compared to neural network models generated by TinyNAS using magnitude-based weight pruning (Zhu et al., 2017) to fit the memory constraint for ESP32.

**Hypothesis 3.** An ML model developed with a majority of control participants will achieve similar Sensitivity and Specificity for participants with lower limb amputation compared to control participants.

## 2. Methods

An experimental human subject protocol was performed to provide laboratory-based examples of activities of daily living (ADLs) and falling and provide the basis for the ML models. Next, an AutoML pipeline was developed to process and optimize the model creation process.

### 2.1. Experimental data collection

A total of 35 participants were recruited, comprising 30 control participants with no lower limb amputation (21 females, 9 males; mean age: $28.3 \pm 8.5$ years) and 5 individuals with lower limb amputation (1 female, 4 males; mean age: $43.0 \pm 13.5$ years). Among the amputee group, 60% ($n = 3$) had transfemoral (above-knee) amputations and 40% ($n = 2$) had transtibial (below-knee) amputations. The overall cohort consisted of 63% female and 37% male participants. The experimental human subject protocol is provided in Appendix A, with Table A1 presenting detailed demographic information for each participant.

IMU sensors (XSens, Enschede, N–etherlands, 100 Hz recording) were placed proximally on the shanks of participants (Fig. 1). In a laboratory setting, participants performed ADLs (walking, running, turning, sitting/standing from a chair, lying down/rising from a bed, picking an item from the floor, and ascending/descending stairs and inclines) as well as simulated falls (forward, backward, left/right lateral, trip and recovery). All activities of daily living were grouped with the label 'ADL' and all fall types were grouped as 'Fall' during model training for the FDS.

### 2.2. MicroNAS development

Details on data division, model selection, data segmentation, creation of MicroNAS and its functions, and ML data processing are presented below.

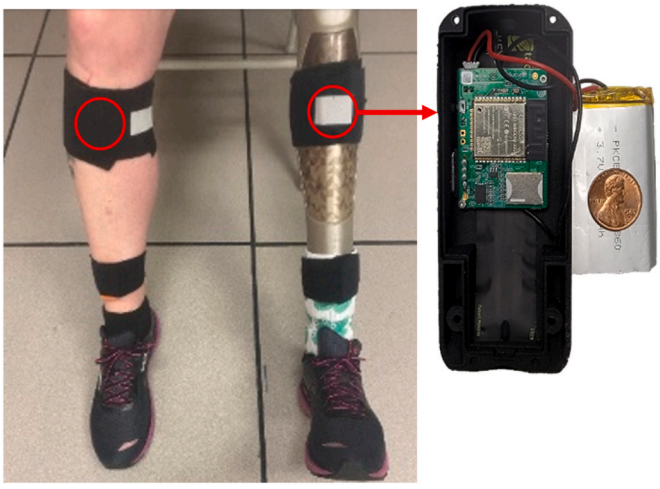

**Fig. 1.** IMU sensors mounted on both shanks and on the prosthetic pylon of a lower-limb amputee; red circles indicate sensor locations. The embedded system on the right is Espressif ESP32 32-bit SoC. The ESP32 offers low cost, low power consumption, and a compact form factor, making it suitable for integration into prosthetic devices. Its memory capacity (320 KB) guided model development within our AutoML framework. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

#### 2.2.1. Data division

Data division created different Training, Validation, and Test datasets by shuffling data from each participant. The Test set consisted of a single participant. Each participant was in the Test set once creating 35 unique test datasets for evaluating model performance and statistical testing. The remaining participants were then divided into Training and Validation sets (90% and 10% of the remaining data, respectively) (Tao et al., 2020). For each of the 35 combinations, data from the single Test participant were first removed, using leave-one-participant out (Vehtari et al., 2017). The Validation set always consisted of three amputee participants and three control participants, chosen using Uniform Randomization from each group (Python 3.10 random library). The remaining participant data was then used for Training (Fig. 2). Training data included IMU signals from both shank sensors, boosting data volume. Data for Validation and Test only included from the sensor on the prosthetic limb for amputees and the non-dominant limb for control participants.

#### 2.2.2. Model selection

Falling happens infrequently compared to daily activities, with this dataset containing 98.3% ADL and only 1.7% Fall classes. We first introduce the class-imbalance handling methods incorporated into the neural networks, followed by a description of the baseline models used for comparison and statistical analysis.

*2.2.2.1. Class imbalance handling.* **Weighted loss:** We employed a weighting method (King et al., 2001) that assigns higher weights to the fall class, with weights ($w_i$) defined as:

$$w_i = \frac{n}{kn_i} \tag{1}$$

where the total number of windows (section 2.2.3) ($n$), the number of windows in each class ($n_i$), and the number of classes ($k$) (in our case $k$ is 2) determine the weight ($w_i$). The weight for the minority Fall class is approximately 29.41 ($\frac{1}{2\times 0.017}$), while the weight for the majority ADL class is approximately 0.51 ($\frac{1}{2\times 0.983}$).

These weights are then considered in binary Cross-Entropy loss during the training phase. Cross-entropy loss is a commonly used loss function in classification problems that calculates the difference between the predicted probability distribution and the actual probability distribution of a set of classes (Fall versus ADL). During the backward propagation phase of the training process, the gradient of the binary cross-entropy loss with respect to the model's parameters (weights and biases) is calculated and used to update the model's parameters. Weighted cross entropy corresponds to:

$$CE = w_1 t_1 \log(f(s)_1) + w_2 t_2 \log(f(s)_2), \tag{2}$$

where $w_1$ and $w_2$ represent the weight assigned to each class ($i$) to account for class imbalance, $t_1$ and $t_2$ represent the true label for class 1 and 2, and $f(s)_1$ and $f(s)_2$ represent the predicted probability or score assigned by the model to class 1 and 2 for a particular input sample $s$.

**Focal loss:** We also implemented Binary Focal Cross-entropy (Lin et al., 2017) for addressing class imbalance. This loss function introduces a modulating factor that down-weights well-classified 'Easy' samples to focus the model's learning on 'Hard' minority samples (falls) (Lin et al., 2017). The loss is defined as:

$$FL(p_t) = -\alpha(1-p_t)^{\gamma}\log(p_t), \tag{3}$$

where $p_t$ represents the predicted probability of the correct class, and $\gamma$ is the focusing parameter. When $\gamma = 0$, focal loss reduces to standard cross-entropy loss; as $\gamma$ increases, the loss contribution from easy-to-classify ADL samples is significantly reduced. For example, with $\gamma = 2$, a correctly identified ADL sample with 90% confidence ($p_t = 0.9$) has its loss contribution reduced by a factor of 100 ($(1-0.9)^2 = 0.01$), preventing the majority class from disproportionately influencing the gradient during training. Hyperparameters $\alpha$ (balancing parameter) and $\gamma$ (focusing parameter) are optimized in the ranges of [0.1, 0.9] and [0.0, 5.0], respectively.

**Neural network models:** Neural network models studied in our framework included a 1D Convolutional Neural Network (1D CNN) (Risso et al., 2022) and a Gated Recurrent Unit (GRU) (Ma et al., 2022), both efficient in handling time series data. 1D CNN and GRU transform the representation of input data by feature extraction and make predictions based on features. 1D CNNs utilize filters to conduct convolution operations that capture temporal hierarchies in the segmented data (Section 2.2.3). GRUs are equipped with gates that regulate information flow, enabling them to capture temporal dependencies and retain past information in the segmented data.

**Baseline models:** Ensemble models designed to address class imbalance, RUSboost (Seiffert et al., 2009) and EasyEnsemble (Liu, 2009), were selected. The ensemble models' performance was compared to neural networks (Hypothesis 1). Finally, H2O-AutoML was selected as the baseline for comparing our developed models. H2O-AutoML is an ensemble of various ML models and outperforms other AutoML tools on similar problems (Truong et al., 2019).

**Statistical analysis:** A comparative analysis was conducted to evaluate whether focal loss provides a performance advantage over the weighted cross-entropy method. A pairwise Wilcoxon statistical test across the 35 test sets (Section 2.2.1) was performed to compare the 1D-CNN model trained with focal loss against the same model trained with

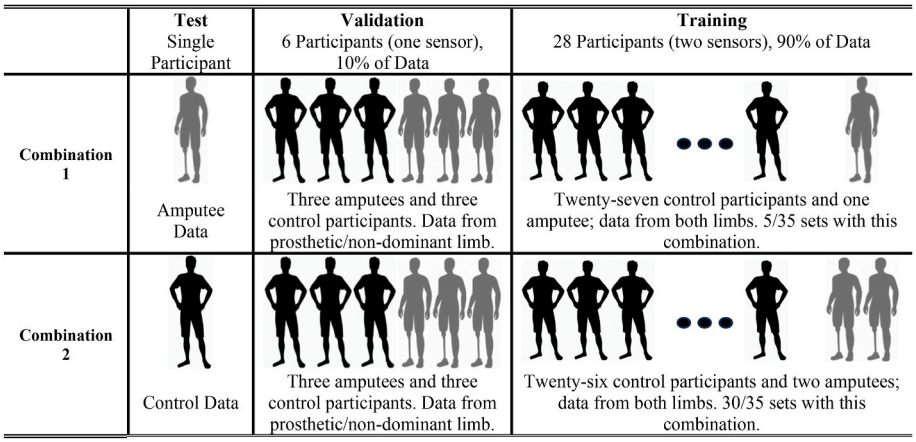


**Fig. 2.** Data division into Training, Validation, and Test sets. The Test set consisted of a single participant with a lower limb amputation (Combination 1) or no lower limb amputation (Combination 2). Each participant was in the Test set once creating 35 unique test datasets for evaluating model performance and statistical testing.

weighted cross-entropy loss. A similar analysis was conducted for the GRU model using both loss functions. The loss function that yields the higher F1-score for the fall class was selected for subsequent hypothesis testing (Section 2.3).

#### 2.2.3. Data segmentation

For neural network models, we segmented the raw data (3 axis accelerometer and gyroscope from the IMU) into sliding windows (Dehghani et al., 2019) to capture the dynamic nature of activities (Fig. 3). Fall duration was approximately 1 s across all participants and pre-fall motions can often indicate an incoming fall (Hemmatpour et al., 2019), so we considered a window size of 1.2 s (120 samples at the 100 Hz sampling rate). We considered 90% overlap (10% step) for consecutive windows, which corresponds to a step size of 0.12 s (12 samples). This specific windowing strategy was chosen to identify the shifts between activities and provide large volume data for the neural network's training phase.

#### 2.2.4. MicroNAS

We designed MicroNAS, a neural architecture search tool, with a focus on identifying high performance (F1-score) architectures tailored for memory-constrained microcontrollers. MicroNAS development is organized into two key subsections: Algorithm and Search Space.

#### 2.2.5. Algorithm

Fig. 4 outlines the flowchart of MicroNAS. MicroNAS utilizes the model size and the search space as inputs, creating models that can be deployed on microcontrollers (e.g., the ESP32 hardware with 320 KB runtime memory sizes). The model size refers to the static model footprint, calculated by multiplying the total number of parameters (weights and biases) by 4 bytes (32-bit floating-point precision). MicroNAS treats the ESP32's 320 KB SRAM limit as a hard constraint for the model size, strictly enforcing a maximum capacity of 81,920 floats (327,680 bytes ÷ 4 bytes per float) during the architecture search process. This conservative constraint ensures the model architecture remains compact enough to facilitate efficient memory management and activation buffering within the limited SRAM. Random Search (Li et al., 2020) maximizes the objective function (F1-score) by simultaneously exploring various layer sequences and hyperparameters (Section 2.2.4.2 Tables 1 and 2). Keras Tuner (Joshi et al., 2021) in Python, a library developed for tuning hyperparameters of Neural network models, was used to implement the Random Search. A preliminary comparison of Random Search against Bayesian Optimization and a Genetic Algorithm demonstrated its suitability for this study, as it provided comparable predictive performance with lower computational overhead (Appendix F). Once Random Search determines an architecture with specific hyperparameters that meets the memory constraint, Adaptive Moment Estimation (Adam) (Kingma and Jimmy, 2014) iteratively minimizes the cross-entropy loss function of the determined architecture. Adam utilizes an extension of stochastic gradient descent with backward propagation (Parashar et al., 2023) to update the architecture's parameters, including weights and biases, over a series of epochs. We adopted a fixed-budget approach (Hansen et al., 2016), developing 20 architectures that meet the memory constraint as our termination criterion. This strategy was implemented to both provide our models with opportunities for enhancing their F1-scores and ensure the completion of testing our three hypotheses within a month, on available hardware. The execution took place on Tempest, a high-performance computing research cluster at Montana State University, utilizing 25 GB of RAM and a single Nvidia A100 GPU.

*2.2.5.1. Search space.* Categorized into two areas: 1) architecture structure (layer sequences) and 2) hyperparameters. MicroNAS defines a three-dimensional search space (depth, width, and temporal resolution (Tan et al., 2019)) with diverse layer combinations, to provide the largest search space possible to find a memory-efficient architecture for the time series data.

**Architecture structure (layer sequences):** Table 1 presents the order of different layer types in our 1D CNN and GRU specific pipelines. Each pipeline begins with a batch normalization layer. In the 1D CNN pipeline, the first block comprises a convolutional layer and optional batch normalization and pooling layers. A second block, identical to the first, is optional and can be repeated based on the microcontroller's memory constraint (up to 10 times allowing the creation of networks with varying depths). Preceding the final block, there is an optional selection of global average pooling, dropout layer, and a fixed flatten layer. The last block (can be repeated up to 3 times) includes optional batch normalization layer, dropout layer, and fully connected layers with different number of neurons for dropout and fully connected (Table 2) allowing the creation of networks with varying widths. The GRU pipeline resembles the 1D CNN pipeline, with the distinction that it incorporates a GRU layer instead of a convolutional layer. The described layer sequences facilitate the exploration of various architecture structures including combinations of Convolution, Pooling, Dropout, Batch Normalization similar to previous human activity recognition systems (Ramanujam et al., 2021b; Chen and Yang, 2015; Ronao et al., 2016; Wan et al., 2020; Shojaedini et al., 2020).

**Hyperparameters:** Table 2 details the hyperparameters utilized by MicroNAS. Hyperparameter base ranges were determined by existing literature to encompass the widest range of useable space.

#### 2.2.6. Ensemble data processing

RUSBoost and EasyEnsemble are ensemble models that provide a baseline for comparison with the MicroNAS developed neural network models. For ML model development, Random Search tuned the hyperparameters listed in Table 3. The ensemble models were trained on raw data in the Training set. In the Validation and Test set, we utilized a majority vote approach, involving the smoothing (Pew et al., 2017) of Fall/ADL predictions across 120 predictions to improve the F1-score of predictions. Ensemble models make predictions for 120 samples (using majority vote on a window) with 90% overlap for the next predictions (Fig. 5) to make them comparable with the neural network models (Fig. 3).

#### 2.2.7. Hardware profiling

To validate the deployability of MicroNAS models, we performed hardware-specific profiling for the ESP32. Models were compiled using the Edge Impulse EON Compiler (Reddi et al., 2023). Inference latency, peak RAM usage, and Flash memory were estimated using a cycle-accurate, hardware-aware simulation. Energy consumption $E$ was calculated as:

$$E = V \times I \times t \tag{4}$$

where $V = 3.3\,\text{V}$, $I = 68\,\text{mA}$ (average active current during CPU computation per ESP32-S2 datasheet[2]), and $t$ is the inference time. Due to current compiler limitations for GRU within the target platform's optimization stack, hardware profiling was performed for the 1D-CNN as the representative architecture for deployment feasibility.

### 2.3. Hypothesis testing

We considered the F1-score of positive class (fall) as our optimization metric and for our hypothesis testing, as we were solely interested in the performance of the fall detector. F1-score is the harmonic mean of precision (which reduces false alarms) and recall (which increases fall detection). F1-score provides a single metric to evaluate the performance of neural networks and ensemble models focused on the minority class (Desuky and Sadiq, 2021).

[2] https://documentation.espressif.com/esp32_datasheet_en.pdf.

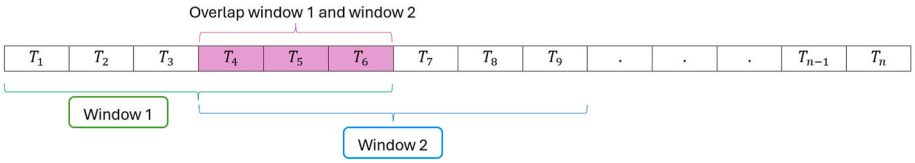


**Fig. 3.** Windowing with overlap. T indicates the time stamp where T1 is the sample recorded at 0.01 s and T2 at 0.02 s during 100 Hz recording (Figure does not show time windows to scale).

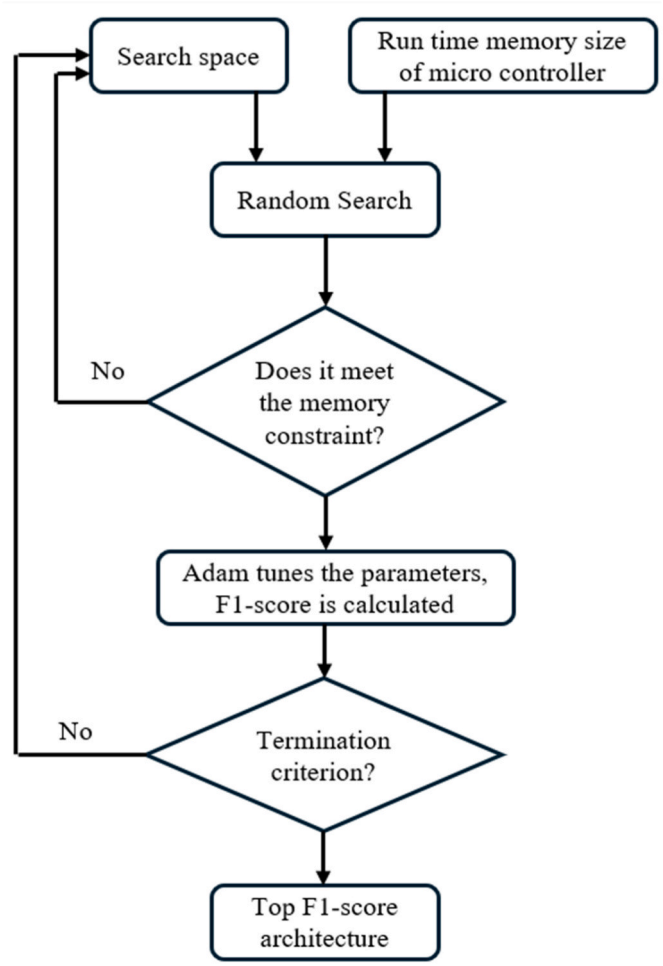


**Fig. 4.** Model development and stages in MicroNAS.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

Precision is the ratio of correctly predicted positive windows (Falls) to the total predicted positives. Precision determines the accuracy of the fall predictions, specifically, it measures the proportion of windows that were correctly identified as falls out of all windows predicted as falls.

$$Precision = \frac{True\ Pos}{True\ Pos + False\ Pos} \tag{6}$$

Recall is the ratio of correctly predicted positive windows to all actual windows in the positive class. It measures the ability to identify falls out of all actual falls.

$$Recall\ (Sensitivity) = \frac{True\ Pos}{True\ Pos + False\ Neg} \tag{7}$$

Specificity measures the model's ability to correctly identify actual negative windows (ADL). It represents the proportion of non-fall activities that were correctly classified as such, which is critical for minimizing false alarms in real-world applications.

$$Specificity = \frac{True\ Neg}{True\ Neg + False\ Pos} \tag{8}$$

Low precision results in high false alarms and low recall indicates missing actual falls. While Precision measures the reliability of a fall trigger, Specificity provides insight into how well the model filters out daily activities that could be mistaken for falls. In addition, the macro-averaged precision, recall, and F1-score were reported to reflect the overall performance of our models. Macro-averaging treats each class (Fall/ADL) equally by calculating the metric for each class independently and then taking the average (Berger et al., 2020). Macro-averaging considers ML performance in both Fall and ADL classes, irrespective of their frequencies.

Hypothesis 1: Neural network models optimized with MicroNAS achieve higher performance, based on F1-score, compared to ensemble models or open-source AutoML tools capable of handling class imbalance automatically and operating with minimal memory requirements for ESP32.

Hypothesis 1 compares the F1-scores of two neural network models (1D CNN and GRU), two ensemble models (RUSBoost and Easy-Ensemble), and one open-source AutoML method (H2O AutoML) with each method generating an F1-score for each participant. Comparisons were made between models to determine differences. Parametric tests (ANOVA, *t*-test) have assumptions of Normality, equal variances, and independence. The Shapiro-Wilk test (Mudholkar et al., 1995) was used to evaluate if the 35 F1-scores followed a normal distribution. The Levene's test (O'Neill et al., 2002) was used to assess the equality of variances. Since F1-scores are generated by different ML models, the independence assumption is validated. The Shapiro-Wilk test indicated that the F1-score samples for our models did not follow a normal distribution. Therefore, we used the Kruskal-Wallis test (McKight et al., 2010), a nonparametric version of ANOVA followed by the Wilcoxon

**Table 1**
Architecture (flow of data through the network) of developed pipelines.

| Block Name | Layer type | Option | Range |
|---|---|---|---|
| | Batch normalization | Yes | 1 |
| CNN or GRU | Convolution or GRU | Yes | Append or remove random in the range of (Miller et al., 2001; Rastogi et al., 2021) |
| | Batch normalization | Yes or no | |
| | Pooling (CNN Only) | Yes or no | |
| Pre-final | Global average pooling | Yes or no | 1 |
| | Dropout layer | Yes or no | |
| | Flatten layer | Yes | |
| Final | Batch Normalization | Yes or no | Append or remove random in the range of (Miller et al., 2001; Terroso et al., 2014) |
| | Fully Connected | Yes or no | |
| | Dropout | Yes or no | |
| | Fully Connected | Yes | 1 |

**Table 2**
Selected hyperparameters for tuning in neural network models. The information within parentheses indicates that a specific hyperparameter is applicable to a particular model architecture.

| Hyperparameter | Definition | Purpose | Value | References |
|---|---|---|---|---|
| Number of Filters (CNN) | Number of filters in a convolutional layer | Controls model depth and complexity. More filters increase complexity and extract additional features. Related to width of the search space. | 1~500 | (Hellmers et al., 2023) ( Mekruksavanich et al., 2023) ( Napieralski et al., 2022) (Lu et al., 2020) (Liberis et al., 2021) |
| Filter Size (CNN) | Size of the filters in a convolutional layer | Determines feature extraction's receptive field. Related to temporal resolution of search space. | 1~ 8 | (Hellmers et al., 2023) ( Mekruksavanich et al., 2023) ( Napieralski et al., 2022) (Lu et al., 2020) (Liberis et al., 2021) |
| Filter Stride (CNN) | Number of samples the filter omits during input convolution | Reduced memory utilization and increased inference computation | 1,2 | (Mekruksavanich et al., 2023) ( Liberis et al., 2021) |
| Padding | Number of zero value time steps added to each side of the input data. Applied to input data before convolution. | Preserves spatial dimensions and prevents information loss at the edges of windows to increase performance. | Yes, no | Li et al. (2022) |
| Activation Function | Applied to the output of a neuron. Rectified Linear Unit function, hyperbolic tangent, and sigmoid activation functions. | Enables non-linear learning | ReLU, Tanh, Sigmoid | (Avilés-Cruz et al., 2019) (Lin et al., 2020b) (Bangaru et al., 2021) |
| Pooling (CNN) | Down sampling operation to reduce spatial dimensions of an input data by returning the maximum or average of the arrays of input data. | Reduces computation and extracts features at different scales. Improves memory efficiency. | Pool length: 2, 4,8,16<br>Max or Average | (Mekruksavanich et al., 2023) ( Avilés-Cruz et al., 2019) (Lu et al., 2020) |
| GRU size (GRU) | Dimensionality of the learned representation influences the model's ability to capture and learn patterns in sequential data | Determines the dimensionality of the output space | 30~256 | (Hellmers et al., 2023) (Lu et al., 2020) |
| Number of Neurons in Each Fully connected Layer | Connects neurons from the previous layer to the current layer. | More neurons increase model complexity and memory requirement | 3~1024 | (Lin et al., 2020b) (Hellmers et al., 2023) (Mekruksavanich et al., 2023) (Napieralski et al., 2022) (Li et al., 2022) |
| Batch size | Number of windows that are processed together | Improves computational efficiency in the training phase. | 916 | (Avilés-Cruz et al., 2019) (Lin et al., 2020b) (Mekruksavanich et al., 2023) (Géron, 2022) |
| Epoch | One epoch is when the model processes all the batches in the training phase. | Ensures that the neural network learns from the entire training dataset in an iterative manner. For this dataset, 100 is effectively no upper limit. | 100 | Kim et al. (2025) |
| Early stopping | If the Cross-Entry loss does not decrease between a set number of epochs, training was terminated. The models is saved at the point of the last Cross-Entropy loss decrease. | Prevents overfitting and improves generalization to unseen data. | 10 | Lu et al. (2020) |
| Dropout | Temporarily ignoring neurons (excluding output neurons) during training for distribution of weighting. | | P = 0.2~0.3 (GRU)<br>P = 0.4~0.5 (CNN) | (Mekruksavanich et al., 2023) ( Napieralski et al., 2022) (Li et al., 2022) (Géron, 2022) |
| Initial Learning Rate | Starting magnitude for the optimization step of the stochastic gradient descent algorithm which influences convergence speed. | | 0.000001~0.01 | (Avilés-Cruz et al., 2019) (Hellmers et al., 2023) (Lin et al., 2020b) (Choi et al., 2022) (Passos et al., 2022) |
| Lasso Regularization | Adds a penalty term to the Cross-Entropy loss function. Encouraging sparsity in the weights of the dense layers. | | 10-5 ~ 10-3 | Kim (2020) |
| Optimizer | Adaptive Moment Estimation (Adam) is an extension of stochastic gradient descent. It is selected due to its good convergence and speed quality | Tunes the parameters (weights and biases) of the neural network | Adam | (Géron, 2022) (Passos et al., 2022) ( Mekruksavanich et al., 2023) |

Signed-Rank test (Rosner et al., 2006), a non-parametric test, with Bonferroni correction to determine differences between the five models tested.

Hypothesis 2**:** Neural network model architectures generated by MicroNAS, without pruning, will achieve higher F1-scores and lower memory sizes compared to neural network models generated by TinyNAS using magnitude-based weight pruning (Zhu et al., 2017), to fit the memory constraint for ESP32.

TinyNAS follows a two-stage process in which a high-performance model is first identified and subsequently pruned to meet memory constraints. We emulated this workflow by initially performing neural architecture search without memory limits to obtain the highest F1-score model, and then iteratively pruned the architecture until it satisfied the 320 KB runtime memory constraint of the ESP32. We implemented this two-stage search-then-prune strategy within our own pipeline rather than directly using the TinyNAS implementation for two reasons. First, TinyNAS is primarily designed for 2D image-based architectures, whereas our application involves multivariate 1D IMU time-series data. Second, TinyNAS optimizes FLOPs during the first stage, while our objective is to maximize the F1-score of the fall class. Incorporating the two-stage methodology into our pipeline ensured identical data preprocessing, class-imbalance handling, and optimization metrics across models, thereby isolating the optimization strategy (single-stage versus two-stage) as the only variable under comparison. This design also increased the likelihood that the baseline method would perform competitively.

We selected 1D CNN as its hyperparameters (number of filters, filter size, stride size, and pooling size) provide more flexibility compared to GRU for pruning and our hypothesis testing. MicroNAS considers the memory of the microcontroller from the beginning. In contrast, TinyNAS does not consider the constraint on memory and focuses on finding the highest F1-score model in stage 1. Then in stage 2, it prunes the model to fit the memory of ESP32. The Wilcoxon signed rank test compares the F1-scores and memory sizes of the 1D CNN models from MicroNAS and

**Table 3**
Selected hyperparameters for tuning in Ensemble models.

| Hyperparameter | Definition | Purpose | Values | References |
|---|---|---|---|---|
| Number of estimators | The number of trees in the forest. | Reduces overfitting and improve F1-score. | (10,250) | (Hellmers et al., 2023) ( Hasan et al., 2022) ( Brajesh et al., 2020) |
| Max depth | The maximum depth of each decision tree. | | (2,46) | (Hellmers et al., 2023) ( Brajesh et al., 2020) |
| Learning rate | The rate at which the model learns from the data. | | (0.001,1) | (Hellmers et al., 2023) ( Hasan et al., 2022) |
| Max features | The maximum number of features each tree is allowed to use. | | ["sqrt", "log2", None] | |

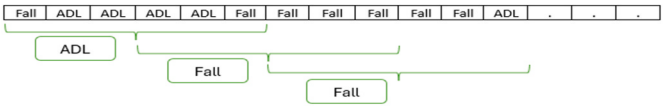


**Fig. 5.** Windowing on Ensemble model predictions. Each prediction is for each time step and majority voting is used on 120 consecutive predictions.

pruned 1D CNN created by TinyNAS.

**TinyNAS pruning setup:** Initially we started with imposing no constraint on memory for TinyNAS and iteratively reduced the memory size (up to 2400 KB) until magnitude-based weight pruning could create a model that fit the memory of ESP32. Pruning reduces the memory size of our model by gradually reducing the number of parameters. It utilizes initial and final sparsity as hyperparameters in an optimization loop to incrementally remove less important weights during training. Sparsity refers to the property where a subset of the model parameters (weights) have a value of exactly zero. Initial sparsity is the fraction of weights set to zero at the beginning of the pruning process. The weights with the smallest absolute values are usually the first to be pruned because they have the least effect on the network's output. Final sparsity is the targeted fraction of weights to be zero by the end of the pruning process. If the pruned model does not fit into the 320 KB memory of the ESP32, the values for initial and final sparsity are iteratively modified in 10% increments until the final pruned model fits the microcontroller's memory.

Hypothesis 3**:** An ML model developed with a majority of control participants will achieve similar Sensitivity and Specificity for participants with lower limb amputation compared to control participants.

A binomial generalized linear mixed model (GLMM) (Bates et al., 2015) was utilized for statistical analysis (Appendix B). The analysis was performed for the GRU model created by MicroNAS.

## 3. Results

The Wilcoxon statistical test revealed no significant difference in F1-score performance between paired models ($p > 0.05$; Table A2 in the Appendix). Given the equivalent performance and the lower computational complexity of implementation, weighted cross-entropy was utilized as the standard loss function for all subsequent hypothesis testing.

Hypothesis 1: Average F1-scores for the models are illustrated in Fig. 6. The 1D CNN ($0.64 \pm 0.17$) and GRU ($0.66 \pm 0.18$) classifiers outperformed the RUSBoost ($0.44 \pm 0.13$), EasyEnsemble ($0.44 \pm 0.13$), and H2O-AutoML ($0.22 \pm 0.07$) models in terms of F1-scores ($p < 0.01$). Ten pairwise comparisons were made to include all combinations between each model (Table 4). Both 1D CNN and GRU outperformed RUSBoost, EasyEnsemble, and H2O-AutoML ($p < 0.01$), supporting our first hypothesis. Similarly, RUSBoost and EasyEnsemble outperformed the baseline H2O-AutoML ($p < 0.01$). The statistical testing showed that models belonging to the same category—neural networks (1D CNN and GRU, $p = 0.42$) and ensemble models (RUSBoost and EasyEnsemble, $p = 0.48$)—had statistically similar performance.

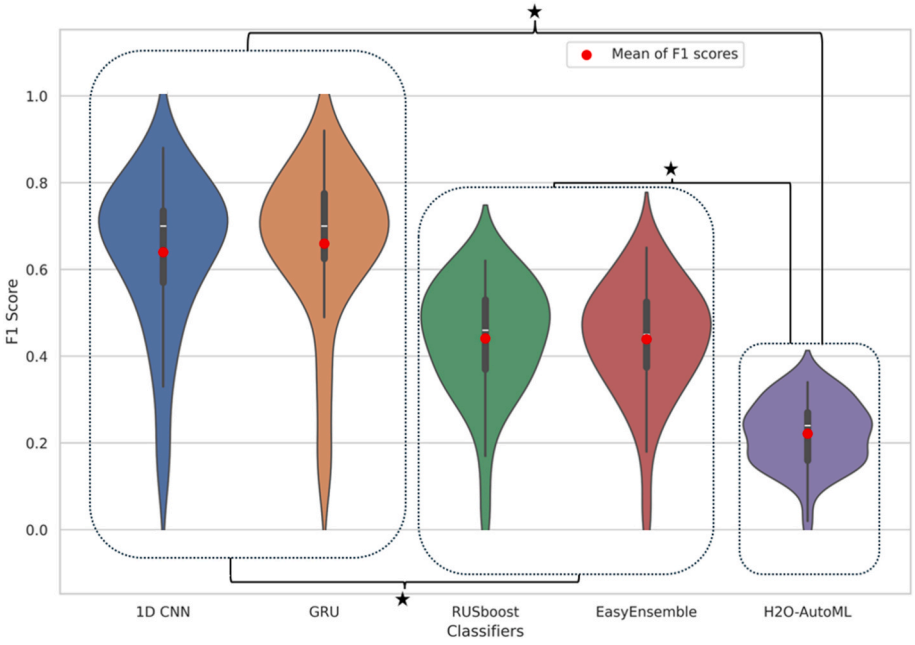


**Fig. 6.** Comparison of Fall class F1-scores for five different models. Dotted lines distinguish models belonging to the same group: neural networks, ensembles, and the baseline H2O-AutoML. The red dot represents the mean, and the violin plots represent the F1-scores of the fall class across the 35 participants listed in Table A2. A star denotes a significant difference ($p < 0.01$) between the corresponding groups. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

**Table 4**
Details of ML model comparisons for three hypotheses. Bolded entries indicate statistically significant differences.

| Hypothesis | Group 1 | Group 2 | P-value |
|---|---|---|---|
| H1 | 1D CNN | GRU | 0.42 |
| | 1D CNN | RUSBoost | **<0.01** |
| | 1D CNN | EasyEnsemble | **<0.01** |
| | 1D CNN | H2O-AutoML | **<0.01** |
| | GRU | RUSBoost | **<0.01** |
| | GRU | EasyEnsemble | **<0.01** |
| | GRU | H2O-AutoML | **<0.01** |
| | RUSBoost | EasyEnsemble | 0.48 |
| | RUSBoost | H2O-AutoML | **<0.01** |
| | EasyEnsemble | H2O-AutoML | **<0.01** |
| | 1D CNN (memory) | GRU (memory) | 0.06 |
| H2 | 1D CNN (memory) | Pruned 1D CNN (memory) | **<0.01** |
| | 1D CNN | Pruned 1D CNN | 0.14 |
| (Sensitivity) | GRU (amputee) | GRU (control) | 0.236 |
| H3 (Specificity) | GRU (amputee) | GRU (control) | <0.01 |

Hypothesis 2: The Wilcoxon signed-rank test indicated that the F1-scores of the 1D CNN and the pruned 1D CNN used to emulate the TinyNAS workflow were statistically comparable (Table 4, $p = 0.14$; Fig. 7, left). However, the 1D CNN had lower memory sizes compared to the pruned 1D CNN (Table 4, $p < 0.01$; Fig. 7, right).

Hardware-specific performance estimation for the 1D CNN (Fig. 8) on the ESP32 indicated an inference latency of 299 ms and a peak RAM usage of 86.8 KB. This occupies only 27% of the available 320 KB SRAM, leaving ample resources for system overhead. Additionally, the model requires 245.1 KB of Flash memory and consumes approximately 67.1 mJ per inference.

A full report of the Macro F1-score, Precision, Macro Precision, Recall, and Macro Recall values is provided in Table A3 in the Appendix.

Hypothesis 3: The GLMM model estimates that the control group has a mean odds ratio of 2.34 for correct fall detection compared to the amputee group (95% CI: 0.57 to 9.55). The estimated probability of correctly detecting a fall (Sensitivity) in the control group is 84.7%, whereas for the amputee group, it is 70.3%. There is weak evidence against the null hypothesis of no difference between the groups ($p = 0.236$). While the difference is not statistically significant, the absolute Sensitivity was lower for the amputee group. A similar analysis was performed to evaluate the Specificity (the ability to correctly identify ADLs). The GLMM modeled the False Positive Rate (probability of a false fall detection) and estimates that the control group has mean odds of a false fall detection that is 2.1 times higher than that of the amputee group (95% CI: 1.43 to 3.08). The False Positive Rate of 0.67% for the control group translates to a Specificity of 99.33%. For the amputee group, the False Positive Rate of 0.3% translates to a Specificity of 99.70%. There is strong evidence that the groups differ ($p < 0.05$).

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Batch Normalization | (120, 6) | 24 |
| Conv1D | (60, 141) | 6,909 |
| Batch Normalization | (60, 141) | 564 |
| Pooling Layer1D | (30, 141) | 0 |
| Conv1D | (15, 31) | 4,402 |
| Pooling Layer1D | (7, 31) | 0 |
| Conv1D | (7, 291) | 27,354 |
| Batch Normalization | (7, 291) | 1,164 |
| Pooling Layer1D | (3, 291) | 0 |
| Global Average Pooling1D | (291) | 0 |
| Flatten | (291) | 0 |
| Batch Normalization | (291) | 1,164 |
| Dense | (1) | 292 |
| | | |
| **Total Parameters** | | **41,873 (163.57 KB)** |
| **Trainable Parameters** | | **40,415 (157.87 KB)** |
| **Non-trainable Parameters** | | **1,458 (5.70 KB)** |

**Fig. 8.** 1D CNN optimal architecture diagram created by MicroNAS.

## 4. Discussion

The objectives of this study were to develop an AutoML framework and demonstrate its application in creating a fall detection algorithm tailored for the lower limb amputee community. This study introduced MicroNAS, an automated neural architecture search tool designed to optimize neural network models for deployment on resource-constrained microcontrollers, specifically the ESP32. The results of our hypothesis testing have provided promising results and highlighted several critical insights.

Hypothesis 1: Neural network models optimized with MicroNAS achieve higher performance, based on F1-score, compared to ensemble models or open-source AutoML tools capable of handling class imbalance automatically and operating with minimal memory requirements for ESP32.

Hypothesis 1 compares the F1-score performance of multiple ML models and aims to determine the top-performing model for the ESP32. F1-scores demonstrated that the MicroNAS models, 1D CNN and GRU, achieved superior performance compared to RUSBoost, EasyEnsemble, and H2O-AutoML (Fig. 6).

The higher F1-score performance of models created by MicroNAS (1D CNN and GRU) compared to other models can be attributed to three factors: 1) how the IMU data is introduced to the models, 2) how the

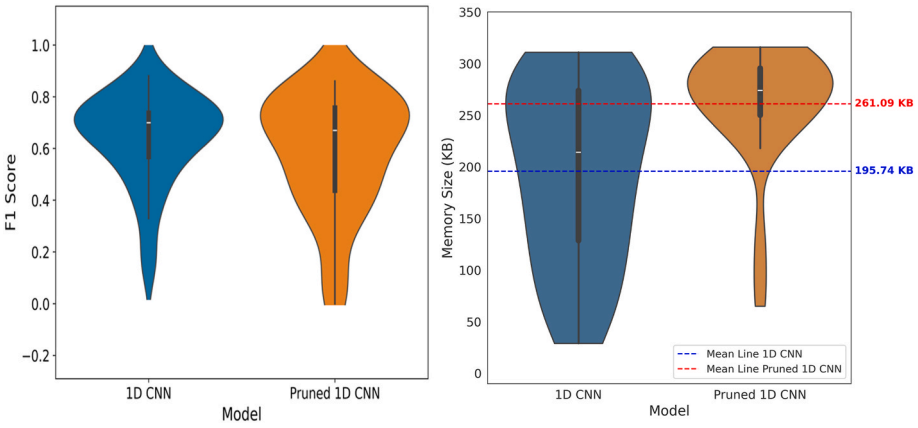


**Fig. 7.** Comparison of Fall class F1-scores (left) and memory sizes (right) for 1D CNN created by MicroNAS and pruned 1D CNN created by TinyNAS.

data is processed, and 3) how class imbalance is handled. MicroNAS receives a window of 120 samples of IMU data, including a three-axis accelerometer and a three-axis gyroscope. It then extracts time-domain features from each window and transforms the representation of IMU data for the classification of falls and ADL. To handle class imbalance, MicroNAS incorporates greater weights for the fall class in the binary cross-entropy loss, ensuring that the misclassification of a fall is more severely penalized, thus affecting the model's parameter updates during the training phase. Consequently, by appropriately introducing the data, transforming its representation, and effectively addressing class imbalance through the weighting method, MicroNAS achieves higher F1-score performance. The high Macro average F1-scores (e.g., $0.82 \pm 0.09$ for 1D CNN) confirm that the model maintains robust performance on the dominant ADL class. Furthermore, the Fall Recall of $0.70 \pm 0.20$ demonstrates that the model successfully prioritizes the minority fall class, preventing the majority class from biasing the optimization process.

Ensemble models receive one sample of IMU data at a time and return the most common class across 120 consecutive predictions (a different method of windowing, Fig. 5). Unlike MicroNAS, ensemble models (RUSBoost and EasyEnsemble) do not extract features from IMU data. Manually extracting features by creating several mathematical functions may not be a feasible approach for deployment on a microcontroller with limited resources. Ensemble models address class imbalance by undersampling ADLs. However, the loss of samples due to undersampling during the training phase reduces the F1-score performance of ensemble models.

Lastly, H2O AutoML receives one sample of IMU data at a time and makes a single prediction for each sample, without taking a majority vote from 120 predictions. To handle class imbalance, H2O AutoML over samples the fall class. However, oversampling can disrupt the temporal dependency between samples in IMU time series data and may introduce synthetic data points for the fall class that are not representative of real fall events, potentially misleading the ML model. Although we used H2O AutoML as prescribed, it was not appropriate for this context. This highlights one of the pitfalls of AutoML—misuse. The misuse of AutoML here demonstrates the importance of human oversight and domain expertise in ensuring that the models generated align with the true characteristics of the data (in our case, time series) and the techniques used for handling class imbalance are effective. The opaque nature of H2O AutoML did not allow us to optimize the model's hyperparameters based on the majority vote of 120 consecutive predictions during the training phase. H2O AutoML's poor F1-score performance underscores the importance of developing a customized AutoML pipeline for IMU data in fall detection, particularly when dealing with imbalanced data. As shown in Fig. 6, as the level of model customization decreases (MicroNAS models, Ensemble models, H2O AutoML), the F1-score performance also declines.

The 1D CNN achieved a memory usage of $195.74 \pm 85.44$ KB. In comparison, the GRU achieved a memory usage of $159.43 \pm 78.81$ KB (Table A2 in the Appendix). The time complexity of 1D CNN in Big O notation is represented by

$$O(dFSCm) \tag{9}$$

where $d$ is the number of convolutional layers, $F$ represents the number of filters in the $i^{th}$ layer, $S$ refers to the spatial dimensions of the filter, $C$ indicates the number of input channels for the $i^{th}$ layer, and $m$ specifies the spatial dimensions of the output feature map (He et al., 2015).

Time complexity of GRU is represented by

$$O(h(d+h)) \tag{10}$$

where $h$ is the number of hidden cells and $d$ is the length of the input sequence (Benoit et al., 2024). An optimal architecture of the 1D CNN and GRU models created by MicroNAS for a randomly selected participant is provided in Figs. 8 and 9.

Although the optimal 1D CNN (41,873 parameters) and GRU (36,021 parameters) share similar model sizes, their inference latency may differ significantly. 1D CNNs process different parts of the input windows in parallel, which may lead to faster inference speeds. This is particularly beneficial for the ESP32, which supports parallel processing with its dual-core CPU. In contrast, GRUs process data sequentially, potentially increasing both battery consumption and inference time. Ultimately, the end user should select the best model for deployment based on the specific needs of the FDS.

Hypothesis 2: Neural network model architectures generated by MicroNAS, without pruning, will achieve higher F1-scores and lower memory sizes compared to neural network models generated by NAS using magnitude-based weight pruning (Zhu et al., 2017), to fit the memory constraint for ESP32.

Hypothesis 2 evaluates the effects of different search spaces and model development approaches on ML performance. The F1-score of the 1D CNN ($0.64 \pm 0.17$) was not statistically different from that of the pruned 1D CNN ($0.58 \pm 0.23$). Similar F1-score performance indicates that expanding the search space during the model development phase does not yield superior models compared to models created by MicroNAS.

MicroNAS creates smaller 1D CNN models ($195.74 \pm 85.44$ KB) compared to the pruned 1D CNN created by TinyNAS ($261.09 \pm 55.95$ KB) ($p < 0.01$, Table 4). MicroNAS tunes 1D CNNs by developing smaller models during the training phase. It fully leverages the weight sharing attribute in 1D CNNs to reduce the number of parameters. MicroNAS utilizes a lower number of filters, reduces filter size, increases filter stride, and applies padding and pooling to create compact CNN models during the training phase.

The advantages of MicroNAS over TinyNAS and other MCU-aware NAS frameworks are summarized as follows:

- MicroNAS produces significantly smaller models than TinyNAS while maintaining comparable F1-score performance.
- MicroNAS develops models in a single stage, reducing computational cost during training, whereas TinyNAS requires a two-stage workflow.
- MicroNAS incorporates L1 regularization to develop sparse models. Although standard dense storage was used in this study, the resulting sparse weight matrices allow end-users to optionally apply lossless compression (e.g., Compressed Sparse Row formats) to further reduce the memory footprint without the performance degradation typically associated with post-training pruning.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Batch Normalization | (120, 6) | 24 |
| GRU | (120, 30) | 3,420 |
| Batch Normalization | (120, 30) | 120 |
| Global Average Pooling1D | (30) | 0 |
| Flatten | (30) | 0 |
| Batch Normalization | (30) | 120 |
| Dropout | (30) | 0 |
| Batch Normalization | (30) | 120 |
| Dropout | (30) | 0 |
| Batch Normalization | (30) | 120 |
| Dense | (1003) | 31,093 |
| Dropout | (1003) | 0 |
| Dense | (1) | 1,004 |
| | | |
| **Total Parameters** | | **36,021 (140.71 KB)** |
| **Trainable Parameters** | | **35,769 (139.72 KB)** |
| **Non-trainable Parameters** | | **252 (1008.00 Byte)** |

**Fig. 9.** Gru optimal architecture diagram created by MicroNAS.

Hypothesis 3: An ML model developed with a majority of control participants will achieve similar Sensitivity and Specificity for participants with lower limb amputation compared to control participants.

Hypothesis 3 examines the generalization performance of the trained model when applied separately to the control population and to individuals with lower-limb amputation. The hypothesis was partially supported. The GRU model achieved significantly higher Specificity for amputee participants compared to controls (99.70% vs 99.33%). However, the Sensitivity was lower for amputees (70.3% vs 84.7%), although this difference was not statistically significant after controlling for subject variance. The findings suggest that, when applied to amputee participants, the model acts more conservatively: it is more likely to miss a fall rather than raise a false alarm compared to its performance on control participants. Future work should address the lower Sensitivity in the amputee population by recruiting a larger sample of individuals with lower-limb amputation and retraining the models prior to clinical deployment.

### 4.1. Comparison to similar works

We acknowledge that comparing our ML models' F1-scores with those in previous studies is not appropriate due to differences in datasets. However, to illustrate the potential advantages offered by MicroNAS, we compare it to a similar FDS that developed deep learning models for pre-impact fall detection on the ARM 32-bit microcontroller (Benoit et al., 2024). The comparative study utilized a sampling frequency of 100 Hz, and used Keras Tuner for the model development. Sampling frequency of 50 Hz can also be used to save battery life (Ajerla et al., 2019). The key differences between this previous study (Benoit et al., 2024) and ours are as follows:

- Sensor placement: The previous study positioned the sensor on the waist. The waist is the steadiest part of the body, showing the least amount of noise and higher accuracy for FDS. In contrast, the thigh (upper shank) demonstrates a higher level of noise and consequently lower accuracy for FDS (Ajerla et al., 2019). We specifically chose to place the IMU on the shank for amputees to integrate it with their prosthesis. Wearing sensors continuously on the body can be uncomfortable, particularly for amputees. To achieve effective real-time fall detection, it is crucial for the amputee to consistently wear the sensor (Jiang et al., 2024), and the sensor on the prosthesis should cause no intrusion.
- Class imbalance: The previous study had 60% fall data (40% ADL), whereas our study had only 1.7% fall samples.
- Memory: The previous study had 512 KB of memory for model creation, whereas we had 320 KB.
- Performance: The previous study provided F1-score results for two participants in test data (1D CNN: 92.84, GRU: 71.92). Our average top F1-scores for two participants (participant 19 and participant A1) are 0.76 for the 1D CNN and 0.89 for the GRU (Table A2 in the Appendix).

### 4.2. Significance

MicroNAS attained significantly higher F1-scores ($0.66 \pm 0.18$) compared to H2O AutoML ($0.22 \pm 0.07$). MicroNAS is an AutoML tool designed to process IMU data and create memory-efficient ML models (1D CNN and GRU) for the ESP32. Its open-source code enables other biomechanists to use and develop FDSs for their desired microcontroller. MicroNAS offers a user-friendly framework: biomechanists input the IMU data and the memory size of their microcontroller, initiate the optimization, and receive an exported model tailored to their microcontroller's memory requirements. Because the exported model is designed to operate directly on the local hardware, the sensor data does not need to be stored in or transmitted to the cloud, allowing the FDS to operate as a private, standalone system.

MicroNAS provides detailed information about various aspects of model development to educate biomechanists on best practices and remove the opaque nature of ML libraries. These aspects include data splitting for human subject studies, data segmentation for IMU time series data, weighting methods and their effects on binary cross-entropy loss to handle class imbalance, search space design for varying architectures, hyperparameter range selection based on previous literature, the role of each hyperparameter, optimizers, and termination criteria based on available computational resources.

MicroNAS bridges the gap between laboratory research and hardware deployment. In many workflows, biomechanists and deployment engineers work independently, often resulting in high-performance models that are too large for target hardware. MicroNAS streamlines this transition by enabling researchers to specify memory constraints directly during the optimization phase. This hardware-aware approach increases the likelihood that models are deployable-by-design, allowing for immediate integration with microcontrollers without requiring the target hardware to be included in the initial optimization loop.

The development of affordable and high-performance FDSs on microcontrollers by biomechanists can provide clinicians with objective insights into a patient's fall history. Clinicians can then use the fall history to develop intervention sessions and prescribe appropriate prosthetic devices.

### 4.3. Limitations

A limitation of the current framework is its reliance on a static footprint (total parameters $\times$ 4 bytes) as a proxy for runtime SRAM usage. While ensuring hardware compatibility, hardware-specific estimations projected a peak RAM usage (86.8 KB) which is below the 320 KB limit. This conservative approach likely over-constrained the search space, potentially excluding more complex architectures that could achieve higher F1-score performance while still fitting within the SRAM through efficient activation buffering. Consequently, these models represent a robust, 'deployable-by-design' baseline rather than the platform's absolute computational ceiling.

Our FDS achieved its best performance with an F1-score of 0.92 with a GRU (Participant 19, Table A2 Appendix). The precision for the fall class was 0.87, and the recall was 0.97. To understand the real-world implications, consider an active individual with lower limb amputation using this FDS for 16 h a day (excluding 8 h of sleep and assuming 2 h in motion for a moderately active individual). In this scenario, the system would generate 936 false alarms daily. Over a prolonged period, if 100 actual falls occur, the FDS would detect 97 of them. To minimize false alarms to just one per day, the FDS would need to achieve an F1-score of approximately 0.99 for the fall class. Fall detection studies typically report metrics such as precision, recall, F1-score, and accuracy. However, it is crucial to focus on the F1-score specifically for the fall class rather than ADL. We achieved a perfect F1-score for the ADL class for participant 19 with GRU, which, although noteworthy, is less important in the context of fall detection.

MicroNAS currently uses the F1-score to select the best architecture during the search phase; however, developers can replace it with the $F_\beta$-score. The parameter $\beta$ controls the trade-off between Precision and Recall. When $\beta > 1$, greater weight is assigned to Recall. When $\beta < 1$, greater weight is assigned to Precision. When $\beta = 1$, the metric reduces to the standard balanced F1-score. By setting $\beta < 1$, for example $F_{0.5}$, the search process places greater emphasis on Precision than Recall, encouraging the selection of architectures that reduce false positives. This trade off may result in some falls going undetected, but it provides clinicians with the flexibility to tune the system based on patient specific risk tolerance, which is an important consideration for practical fall detection system deployment.

The current F1-scores are not sufficient for clinicians and commercial product implementation (Benoit et al., 2024). The current dataset's small size (35 participants) and focus on a single sensor location

(prosthesis pylon), and a high imbalance ratio may limit generalizability. The inclusion of another sensor (e.g., a pressure sensor) or placement at a different location on the prosthesis could potentially increase the F1-score. In addition, we anticipate attaining higher F1-scores for FDS with the availability of a larger dataset for model training. Future work will expand recruitment to include older adults and explore multi-sensor setups. The imbalance ratio was high in our dataset and rare events prediction like falls is inherently challenging. While our proposed weighting method could offset the effect of class imbalance, collecting more fall samples can further improve the FDS performance. Going forward, microcontrollers with larger runtime memory may enable the deployment of DL models with higher F1-scores, which could support clinical adoption.

Our FDS was developed based on laboratory data and the translation to the real world and functional use is still unknown. Age, previous fall experience, and velocity are contributing factors to real falls. The elderly are at a higher risk of falling compared to younger individuals. Moreover, the risk of a fall increases in individuals who have experienced a previous fall. Fall experience influences gait, and individuals at an increased risk of falling tend to walk more slowly (Hemmatpour et al., 2019). Therefore, age and walking speed are influential in the actual risk of falling (which we did not control in this study), and accounting for these factors could likely enhance the performance of FDS in real-world scenarios.

Our laboratory-based fall simulation has fallen short of capturing the full scope of themes that emerged from the lower limb amputee community. A broader characterization of the physical environment (e.g., slippery surfaces, stairs, incline versus decline) and biomechanical factors (perturbations applied to the prosthetic leg during walking and transition activities, varying surface and terrain conditions, performing concurrent cognitive or physical tasks, and a variety of situations (e.g., fatigued, rushed)) is needed to provide a more comprehensive picture of which ground conditions contribute to falls or near-falls in the lower limb amputee population (Kim et al., 2022). In addition, the definition of fall for a lower limb amputee should be considered: A fall is a loss of balance or sudden loss of support where the body lands on the ground, floor, or another object. A near-fall is a loss of balance where one catches oneself or recovers one's balance without landing on the ground, floor, or another object (Ferrell-Olson et al., 2024). In our experimental fall simulation, participants completely landed on a fall pad during fall events. We did not simulate falls or near-falls that involved other objects. Additionally, the element of suddenness or unexpectedness was not fully present in our simulation. While participants wore glasses that blocked their peripheral vision when they hit the fall pad, they were mentally aware that a fall was about to occur.

### 4.4. Future work

In future research, we plan to assess the performance of our FDS, which was developed using laboratory-based data, against data collected from a larger population of lower-limb amputees from diverse age groups in real-world environments. Real-world falls may exhibit different data characteristics, and we aim to see if there is a significant statistical difference between the F1-scores of our model on test data in the lab with the test data in real-world scenarios. Future work will investigate the application of medical domain transfer learning to overcome the challenge of data scarcity in the amputee population. Specifically, this would involve pre-training a base model on large, publicly available IMU datasets with similar sensor placement from control individuals to learn general human kinematic and physical impact features. By freezing the early feature extraction layers and fine tuning the final classification layers using a small dataset from a specific lower limb amputee, the system can be highly personalized to the unique gait mechanics of a prosthesis user without computationally expensive retraining. In addition, we will explore online learning techniques to incrementally adapt the model parameters directly on the microcontroller. In rehabilitation, patient biomechanics change over time due to factors such as muscle strengthening, fatigue, or changes in prosthesis socket fit, a phenomenon known as concept drift. By leveraging a human in the loop approach, for example allowing a user to log a false alarm via a connected smartphone application, the microcontroller can use this real-world feedback to perform on-device weight updates. This adaptation would allow continual improvement to the model's precision while keeping raw medical data securely localized to the user's hardware. Finally, we aim to develop MicroNAS current support and capabilities as follows:

1. **Optimization Algorithms:** The MicroNAS pipeline optimizer currently uses Random Search. We plan to include evolutionary algorithms, such as genetic algorithms and particle swarm optimization, and assess their performance against the built-in optimizers in Keras Tuner, such as Random Search and Bayesian optimization.
2. **Training Time Reduction:** For code execution, we utilized GPU processing; however, the development of our models was time-intensive. The GRU training process is sequential and slower compared to CNNs. To expedite training time for MicroNAS, in the future we plan on using adaptive subset selection (White et al., 2022). Adaptive subset selection trains the MicroNAS models on a dynamically selected subset of the full training data, which is regularly updated during the training process to remain representative of the entire dataset. The objective is to preserve the model's performance while significantly reducing the computational resources required for training.
3. **Handling Class Imbalance:** We used a weighting method for handling class imbalance. Other potential strategies for addressing class imbalance include the generation of synthetic data. While the Synthetic Minority Over-sampling Technique (Chawla et al., 2002) is popular for oversampling, it does not account for the temporal dependency of data. Investigating synthetic data generation techniques that consider temporal dependencies in IMU data (e.g., temporal-aware SMOTE) represents a promising direction for future research.

## 5. Conclusion

FDS development for individuals with lower limb amputations is crucial due to their increased risk of falling and the severe consequences of such falls. This research aimed to overcome the existing limitations in developing an FDS by addressing critical challenges such as the integration of ML techniques suitable for IMU sensors, class imbalance, and memory constraints of current microcontrollers. The contributions of this work are as follows:

1. We collected simulated falls and ADL data from both control and lower limb amputee participants.
2. We developed a neural architecture search tool (MicroNAS) for resource limited microcontrollers. MicroNAS uses the memory size of the microcontroller as a guide for model creation. MicroNAS effectively handles class imbalance and uses AutoML techniques to perform neural architecture search (1D CNN and GRU) for developing an FDS.
3. We demonstrated that MicroNAS saves computational resources by creating models for the ESP32 in a single stage without pruning.

MicroNAS's models surpass the performance of RUSBoost, Easy-Ensemble, and H2O AutoML. MicroNAS neural network models achieved 68% F1-score performance across all participants. The F1-score results were obtained using a single sensor located on the shank with 35 participants. A larger dataset will further refine our current models. Hardware-specific performance estimations indicate that our optimal 1D CNN model utilizes only 86.8 KB of RAM (27% of available memory) with an inference latency of 299 ms on the ESP32. This confirms the

models are deployable on resource-constrained platforms while leaving sufficient headroom for system processes. Biomechanists can utilize MicroNAS's open-source code to train neural network models for microcontrollers with diverse memory capacities.

## CRediT authorship contribution statement

**Seyed Mojtaba Mohasel:** Conceptualization, Methodology, Project administration, Software, Validation, Visualization, Writing – original draft. **John Sheppard:** Formal analysis, Writing – review & editing. **Lindsey K. Molina:** Data curation, Methodology, Resources, Writing – review & editing. **Richard R. Neptune:** Conceptualization, Data curation, Funding acquisition, Project administration, Resources, Supervision, Writing – review & editing. **Shane R. Wurdeman:** Conceptualization, Data curation, Funding acquisition, Resources, Writing – review & editing. **Corey A. Pew:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

## Declaration of competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgment

The authors acknowledge the support of Montana State University's Research Computing Infrastructure (RCI) for providing access to the Tempest high-performance computing cluster. Funding was provided by the CDMRP Grant W81XWH-20-1-0164.

## Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.engappai.2026.116034.

## Data availability

The data that has been used is confidential.

## References

Ajerla, Dharmitha, Mahfuz, Sazia, Zulkernine, Farhana, 2019. A real-time patient monitoring framework for fall detection. Wireless Commun. Mobile Comput. (1), 9507938. https://doi.org/10.1155/2019/9507938, 2019.

Avilés-Cruz, Carlos, Ferreyra-Ramírez, Andrés, Zúñiga-López, Arturo, Villegas-Cortéz, Juan, 2019. Coarse-fine convolutional deep-learning strategy for human activity recognition. Sensors 19 (7), 1556. https://doi.org/10.3390/s19071556.

Babiuch, Marek, Foltýnek, Petr, Smutný, Pavel, 2019. Using the ESP32 microcontroller for data processing. In: 2019 20th International Carpathian Control Conference (ICCC). IEEE, pp. 1–6. https://doi.org/10.1109/CarpathianCC.2019.8765944.

Bangaru, Srikanth Sagar, Wang, Chao, Busam, Sri Aditya, Aghazadeh, Fereydoun, 2021. ANN-based automated scaffold builder activity recognition through wearable EMG and IMU sensors. Autom. ConStruct. 126, 103653. https://doi.org/10.1016/j.autcon.2021.103653.

Bates, Douglas, Mächler, Martin, Bolker, Ben, Walker, Steve, 2015. Fitting linear mixed-effects models using lme4. J. Stat. Software 67, 1–48. https://doi.org/10.18637/JSS.V067.I01.

Benoit, Aurélien, Escriba, Christophe, Gauchard, David, Esteve, Alain, Rossi, Carole, 2024. Analyzing and comparing deep learning models on a ARM 32 bits microcontroller for pre-impact fall detection. IEEE Sens. J. https://doi.org/10.1109/JSEN.2024.3364249.

Berger, Anna, Guda, Sergey, 2020. Threshold optimization for F measure of macro-averaged precision and recall. Pattern Recogn. 102, 107250. https://doi.org/10.1016/j.patcog.2020.107250.

Bourke, Alan K., Lyons, Gerald M., 2008. A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. Med. Eng. Phys. 30 (1), 84–90. https://doi.org/10.1016/j.medengphy.2006.12.001.

Bourke, Alan K., O'brien, J.V., Lyons, Gearid M., 2007. Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm. Gait Posture 26 (2), 194–199. https://doi.org/10.1016/j.gaitpost.2006.09.012.

Brajesh, Sunidhi, Ray, Indraneel, 2020. Ensemble approach for sensor-based human activity recognition. In: Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers, pp. 296–300. https://doi.org/10.1145/3410530.3414352.

Chawla, Nitesh V., Bowyer, Kevin W., Hall, Lawrence O., Philip Kegelmeyer, W., 2002. SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. 16, 321–357. https://doi.org/10.48550/arXiv.1106.1813.

Chen, Jay, Kwong, Karric, Chang, Dennis, Luk, Jerry, Bajcsy, Ruzena, 2006. Wearable sensors for reliable fall detection. In: 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference. IEEE, pp. 3551–3554. https://doi.org/10.1109/IEMBS.2005.1617246.

Chen, Yuqing, Yang, Xue, 2015. A deep learning approach to human activity recognition based on single accelerometer. In: 2015 IEEE International Conference on Systems, Man, and Cybernetics. IEEE, pp. 1488–1492. https://doi.org/10.1109/SMC.2015.263.

Chen, Zhi, Duan, Jiang, Kang, Li, Qiu, Guoping, 2021. Class-imbalanced deep learning via a class-balanced ensemble. IEEE Transact. Neural Networks Learn. Syst. 33 (10), 5626–5640. https://doi.org/10.1109/TNNLS.2021.3071122.

Choi, Ahnryul, Kim, Tae Hyong, Yuhai, Oleksandr, Jeong, Soohwan, Kim, Kyungran, Kim, Hyunggun, Mun, Joung Hwan, 2022. Deep learning-based near-fall detection algorithm for fall risk monitoring system using a single inertial measurement unit. IEEE Trans. Neural Syst. Rehabil. Eng. 30, 2385–2394. https://doi.org/10.1109/TNSRE.2022.3199068.

Cummings, Steven R., Nevitt, Michael C., Sharon, Kidd, 1988. Forgetting falls: the limited accuracy of recall of falls in the elderly. J. Am. Geriatr. Soc. 36 (7), 613–616. https://doi.org/10.1111/j.1532-5415.1988.tb06155.x.

Dauriac, Boris, Bonnet, Xavier, Pillet, Helene, Lavaste, Francois, 2019. Estimation of the walking speed of individuals with transfemoral amputation from a single prosthetic shank-mounted IMU. Proc. IME H J. Eng. Med. 233 (9), 931–937. https://doi.org/10.1177/0954411919858468.

Dehghani, Akbar, Sarbishei, Omid, Glatard, Tristan, Shihab, Emad, 2019. A quantitative comparison of overlapping and non-overlapping sliding windows for human activity recognition using inertial sensors. Sensors 19 (22), 5026. https://doi.org/10.3390/s19225026.

Desuky, Abeer S., Sadiq, Hussain, 2021. An improved hybrid approach for handling class imbalance problem. Arabian J. Sci. Eng. 46, 3853–3864. https://doi.org/10.1007/s13369-021-05347-7.

Ferrell-Olson, Julie, Hafner, Brian J., Sawers, Andrew, 2024. Evaluating fall event definitions relative to lower limb prosthesis users' lived experiences. Disabil. Rehabil. 1–8. https://doi.org/10.1080/09638288.2024.2374501.

Ganz, David A., Higashi, Takahiro, Rubenstein, Laurence Z., 2005. Monitoring falls in cohort studies of community-dwelling older people: effect of the recall interval. J. Am. Geriatr. Soc. 53 (12), 2190–2194. https://doi.org/10.1111/j.1532-5415.2005.00509.x.

Géron, Aurélien, 2022. Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Inc.. ISBN 9781098122461, 1098122461.

Halilaj, Eni, Rajagopal, Apoorva, Fiterau, Madalina, Hicks, Jennifer L., Hastie, Trevor J., Delp, Scott L., 2018. Machine learning in human movement biomechanics: best practices, common pitfalls, and new opportunities. J. Biomech. 81, 1–11. https://doi.org/10.1016/j.jbiomech.2018.09.009.

Hansen, Nikolaus, Auger, Anne, Brockhoff, Dimo, Tušar, Dejan, 2016. Tea tušar. "COCO: performance assessment.". arXiv preprint arXiv:1605.03560. https://doi.org/10.48550/arXiv.1605.03560.

Hasan, Tasnimul, Karim, Md Faiyed Bin, Mahadi, Mahin Khan, Nishat, Mirza Muntasir, Faisal, Fahim, 2022. Employment of ensemble machine learning methods for human activity recognition. J. Healthc. Eng. (1), 6963891. https://doi.org/10.1155/2022/6963891, 2022.

He, Kaiming, Sun, Jian, 2015. Convolutional neural networks at constrained time cost. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5353–5360. https://doi.org/10.48550/arXiv.1412.1710.

He, Xin, Zhao, Kaiyong, Chu, Xiaowen, 2021. AutoML: a survey of the state-of-the-art. Knowl. Base Syst. 212, 106622. https://doi.org/10.1016/j.knosys.2020.106622.

Hellmers, Sandra, Elias, Krey, Gashi, Arber, Koschate, Jessica, Schmidt, Laura, Stuckenschneider, Tim, Hein, Andreas, Zieschang, Tania, 2023. Comparison of machine learning approaches for near-fall-detection with motion sensors. Front. Digit. Health 5, 1223845. https://doi.org/10.3389/fdgth.2023.1223845.

Hemmatpour, Masoud, Ferrero, Renato, Montrucchio, Bartolomeo, Rebaudengo, Maurizio, 2019. A review on fall prediction and prevention system for personal devices: evaluation and experimental results. Adv. Human Computer Interact. (1), 9610567. https://doi.org/10.1155/2019/9610567, 2019.

Hubara, Itay, Courbariaux, Matthieu, Soudry, Daniel, El-Yaniv, Ran, Bengio, Yoshua, 2018. Quantized neural networks: training neural networks with low precision weights and activations. J. Mach. Learn. Res. 18 (187), 1–30. https://doi.org/10.48550/arXiv.1609.07061.

Islam, Md Milon, Omar, Tayan, Islam, Md Repon, Islam, Md Saiful, Nooruddin, Sheikh, Kabir, Muhammad Nomani, Islam, Md Rabiul, 2020. Deep learning based systems developed for fall detection: a review. IEEE Access 8, 166117–166137. https://doi.org/10.1109/ACCESS.2020.3021943.

Jiang, Zhiyuan, Al-Qaness, Mohammed AA., Dalal, AL-Alimi, Ewess, Ahmed A., Abd Elaziz, Mohamed, Dahou, Abdelghani, Helmi, Ahmed M., 2024. Fall detection

systems for internet of medical things based on wearable sensors: a review. IEEE Internet Things J. https://doi.org/10.1109/JIOT.2024.3421336.
Joshi, Sharmad, Owens, Jessie Ann, Shah, Shlok, Munasinghe, Thilanka, 2021. Analysis of preprocessing techniques, Keras tuner, and transfer learning on cloud street image data. In: 2021 IEEE International Conference on Big Data (Big Data). IEEE, pp. 4165–4168. https://doi.org/10.1109/BigData52589.2021.9671878.
Karmaker Santu, Kanti, Shubhra, Mahadi Hassan, Md, Smith, Micah J., Xu, Lei, Zhai, ChengXiang, Veeramachaneni, Kalyan, 2020. Automl to Date and Beyond: Challenges and Opportunities. https://doi.org/10.48550/arXiv.2010.10777 arXiv e-prints : arXiv-2010.
Kim, Byunggun, Kwon, Younghun, 2025. Searching for effective preprocessing method and CNN based architecture with efficient channel attention on speech emotion recognition. Sci. Rep. 15 (1), 32689. https://doi.org/10.1038/s41598-025-19887-7.
Kim, Eunji, 2020. Interpretable and accurate convolutional neural networks for human activity recognition. IEEE Trans. Ind. Inf. 16 (11), 7190–7198. https://doi.org/10.1109/TII.2020.2972628.
Kim, Janis, McDonald, Cody L., Hafner, Brian J., Sawers, Andrew, 2022. Fall-related events in people who are lower limb prosthesis users: the lived experience. Disabil. Rehabil. 44 (15), 3897–3908. https://doi.org/10.1080/09638288.2021.1891467.
King, Gary, Zeng, Langche, 2001. Logistic regression in rare events data. Polit. Anal. 9 (2), 137–163. https://doi.org/10.1093/oxfordjournals.pan.a004868.
Kingma, Diederik P., Jimmy, Ba, 2014. Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980. https://doi.org/10.48550/arXiv.1412.6980.
Li, Congcong, Liu, Minghao, Yan, Xinsheng, Teng, Guifa, 2022. Research on CNN-BiLSTM fall detection algorithm based on improved attention mechanism. Appl. Sci. 12 (19), 9671. https://doi.org/10.3390/app12199671.
Li, Liam, Talwalkar, Ameet, 2020. Random search and reproducibility for neural architecture search. In: Uncertainty in Artificial Intelligence. PMLR, pp. 367–377. https://doi.org/10.48550/arXiv.1902.07638.
Liberis, Edgar, Dudziak, Łukasz, Lane, Nicholas D., 2021. μnas: constrained neural architecture search for microcontrollers. In: Proceedings of the 1st Workshop on Machine Learning and Systems, pp. 70–79. https://doi.org/10.1145/3437984.3458836.
Liebenwein, Lucas, Baykal, Cenk, Carter, Brandon, Gifford, David, Rus, Daniela, 2021. Lost in pruning: the effects of pruning neural networks beyond test accuracy. Proceed. Machine Learn. Syst. 3, 93–138. https://doi.org/10.48550/arXiv.2103.03014.
Lin, Chuan-Bi, Dong, Ziqian, Kuan, Wei-Kai, Huang, Yung-Fa, 2020b. A framework for fall detection based on OpenPose skeleton and LSTM/GRU models. Appl. Sci. 11 (1), 329. https://doi.org/10.3390/app11010329.
Lin, Ji, Chen, Wei-Ming, Lin, Yujun, Gan, Chuang, Han, Song, 2020a. Mcunet: tiny deep learning on iot devices. Adv. Neural Inf. Process. Syst. 33, 11711–11722. https://doi.org/10.48550/arXiv.2007.10319.
Lin, Tsung-Yi, Goyal, Priya, Girshick, Ross, He, Kaiming, Dollár, Piotr, 2017. Focal loss for dense object detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2980–2988. https://doi.org/10.1109/TPAMI.2018.2858826.
Liu, Hangsheng, Chen, Christine, Hanson, Mark, Chaturvedi, Ritika, Mattke, Soeren, Hillestad, Richard John, 2017. Economic Value of Advanced Transfemoral Prosthetics. RAND. https://www.rand.org/pubs/research_reports/RR2096.html.
Liu, Tian-Yu, 2009. Easyensemble and feature selection for imbalance data sets. In: 2009 International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing. IEEE, pp. 517–520. https://doi.org/10.1109/IJCBS.2009.22.
Lu, Huaitian, Lambert, RB Schomaker, Carloni, Raffaella, 2020. IMU-based deep neural networks for locomotor intention prediction. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 4134–4139. https://doi.org/10.1109/IROS45743.2020.9341649.
Ma, Haoyu, Cao, Jianqiu, Mi, Bo, Huang, Darong, Liu, Yang, Li, Shaoqian, 2022. A GRU-Based lightweight system for CAN intrusion detection in real time. Secur. Commun. Network. 1, 5827056. https://doi.org/10.1155/2022/5827056, 2022.
McKight, Patrick E., Najab, Julius, 2010. Kruskal-wallis test. The Corsini Encyclopedia of Psychology. https://doi.org/10.1002/9780470479216.corpsy0491, 1-1.
Mekruksavanich, Sakorn, Jantawong, Ponnipa, Jitpattanakul, Anuchit, 2023. Deep learning approaches for har of daily living activities using imu sensors in smart glasses. In: 2023 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON). IEEE, pp. 474–478. https://doi.org/10.1109/ECTIDAMTNCON57770.
Miller, William C., Speechley, Mark, Deathe, Barry, 2001. The prevalence and risk factors of falling and fear of falling among lower extremity amputees. Arch. Phys. Med. Rehabil. 82 (8), 1031–1037. https://doi.org/10.1053/apmr.2001.24295.
Mohasel, Seyed Mojtaba, Afzal Aghaei, Alireza, Sheppard, John W., Pew, Corey A., 2026c. Automated dual-stream deep network design for activity recognition. J. Biomech. 202, 113290. https://doi.org/10.1016/j.jbiomech.2026.113290.
Mohasel, Seyed Mojtaba, Koosha, Hamidreza, 2026b. A robust and lightweight support vector machine for imbalanced and noisy data via benders decomposition. Neurocomputing, 132629. https://doi.org/10.1016/j.neucom.2026.132629.
Mohasel, SeyedMojtaba, Afzal Aghaei, Alireza, Pew, Corey, 2026a. Initial investigation of Kolmogorov–Arnold networks and fractional deep Learning for personalized prosthetic control: A Pilot Study. IEEE Access. https://doi.org/10.1109/ACCESS.2026.3706104.
Mudholkar, Govind S., Srivastava, Deo Kumar, Thomas Lin, C., 1995. Some p-variate adaptations of the Shapiro-Wilk test of normality. Commun. Stat. Theor. Methods 24 (4), 953–985. https://doi.org/10.1080/03610929508831533.
Napieralski, Jan Andrzej, Tylman, Wojciech, Kotas, Rafał, Marciniak, Paweł, Kamiński, Marek, Janc, Magdalena, Józefowicz-Korczyńska, Magdalena, Zamysłowska-Szmytke, Ewa, 2022. Classification of subjects with balance disorders using 1d-cnn and inertial sensors. IEEE Access 10, 127610–127619. https://doi.org/10.1109/ACCESS.2022.3225521.
O'Neill, Michael E., Mathews, Ky L., 2002. Levene tests of homogeneity of variance for general block and treatment designs. Biometrics 58 (1), 216–224. https://doi.org/10.1111/j.0006-341X.2002.00216.x.
Pandiyan, Manikandan, Stolzman, Jenna, Wooldridge, Margaret S., 2025. Predictive modeling and analysis of industrial flare performance using advanced machine learning approaches. Process Saf. Environ. Prot., 107251 https://doi.org/10.1016/j.psep.2025.107251.
Parashar, Anubha, Parashar, Apoorva, Ding, Weiping, Shekhawat, Rajveer S., Rida, Imad, 2023. Deep learning pipelines for recognition of gait biometrics with covariates: a comprehensive review. Artif. Intell. Rev. 56 (8), 8889–8953. https://doi.org/10.1007/s10462-022-10365-4.
Passos, Dário, Mishra, Puneet, 2022. A tutorial on automatic hyperparameter tuning of deep spectral modelling for regression and classification tasks. Chemometr. Intell. Lab. Syst. 223, 104520. https://doi.org/10.1016/j.chemolab.2022.104520.
Pew, Corey, Klute, Glenn K., 2017. Turn intent detection for control of a lower limb prosthesis. IEEE (Inst. Electr. Electron. Eng.) Trans. Biomed. Eng. 65 (4), 789–796. https://doi.org/10.1109/TBME.2017.2721300.
Radosavovic, Ilija, Johnson, Justin, Xie, Saining, Lo, Wan-Yen, Dollár, Piotr, 2019. On network design spaces for visual recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1882–1890. https://doi.org/10.48550/arXiv.1905.13214.
Radosavovic, Ilija, Prateek Kosaraju, Raj, Girshick, Ross, He, Kaiming, Dollár, Piotr, 2020. Designing network design spaces. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10428–10436. https://doi.org/10.48550/arXiv.2003.13678.
Rainio, Oona, Teuho, Jarmo, Klén, Riku, 2024. Evaluation metrics and statistical tests for machine learning. Sci. Rep. 14 (1), 6086. https://doi.org/10.1038/s41598-024-66611-y.
Ramanujam, Elangovan, Perumal, Thinagaran, Padmavathi, S.J.I.S.J., 2021a. Human activity recognition with smartphone and wearable sensors using deep learning techniques: a review. IEEE Sens. J. 21 (12), 13029–13040. https://doi.org/10.1109/JSEN.2021.3069927.
Ramanujam, Elangovan, Perumal, Thinagaran, Padmavathi, S.J.I.S.J., 2021b. Human activity recognition with smartphone and wearable sensors using deep learning techniques: a review. IEEE Sens. J. 21 (12), 13029–13040. https://doi.org/10.1109/JSEN.2021.3069927.
Rastogi, Shikha, Singh, Jaspreet, 2021. A systematic review on machine learning for fall detection system. Comput. Intell. 37 (2), 951–974. https://doi.org/10.1111/coin.12441.
Ray, Partha Pratim, 2022. A review on TinyML: state-of-the-art and prospects. J. King Saud Univ., Comput. Inf. Sci. 34 (4), 1595–1623. https://doi.org/10.1016/j.jksuci.2021.11.019.
Reddi, Janapa, Vijay, Alexander Elium, Hymel, Shawn, Tischler, David, Situnayake, Daniel, Ward, Carl, Moreau, Louis, et al., 2023. Edge impulse: An mlops platform for tiny machine learning. Proc. Machine Learn. Syst. 5, 254–268. https://doi.org/10.48550/arXiv.2212.03332.
Risso, Matteo, Burrello, Alessio, Conti, Francesco, Lamberti, Lorenzo, Chen, Yukai, Benini, Luca, Macii, Enrico, Poncino, Massimo, Pagliari, Daniele Jahier, 2022. Lightweight neural architecture search for temporal convolutional networks at the edge. IEEE Trans. Comput. 72 (3), 744–758. https://doi.org/10.1109/TC.2022.3177955.
Ronao, Charissa Ann, Cho, Sung-Bae, 2016. Human activity recognition with smartphone sensors using deep learning neural networks. Expert Syst. Appl. 59, 235–244. https://doi.org/10.1016/j.eswa.2016.04.032.
Rosner, Bernard, Glynn, Robert J., Mei-Ling, T. Lee, 2006. The Wilcoxon signed rank test for paired comparisons of clustered data. Biometrics 62 (1), 185–192. https://doi.org/10.1111/j.1541-0420.2005.00389.x.
Sakhaee, Nima, Sarrouf, Stephanie, Kim, Paul, Kim, Jong-Gook, Alshawabkeh, Akram N., 2025. Prediction and optimization of electrochemical oxidation efficiency using machine learning: insights from carbon-based anodes for water treatment. J. Environ. Manag. 394, 127210. https://doi.org/10.1016/j.psep.2025.107251.
Salah, Osama Zaid, Kumar Selvaperumal, Sathish, Abdulla, Raed, 2022. Accelerometer-based elderly fall detection system using edge artificial intelligence architecture. Int. J. Electr. Comput. Eng. 12 (4), 4430–4438. https://doi.org/10.11591/ijece.v12i4.pp4430-4438.
Seiffert, Chris, Khoshgoftaar, Taghi M., Van Hulse, Jason, Napolitano, Amri, 2009. RUSBoost: a hybrid approach to alleviating class imbalance. IEEE Trans. Syst. Man Cybernetics A Syst. Humans 40 (1), 185–197. https://doi.org/10.1109/TSMCA.2009.2029559.
Shawen, Nicholas, Lonini, Luca, Krishna Mummidisetty, Chaithanya, Shparii, Ilona, Albert, Mark V., Kording, Konrad, Jayaraman, Arun, 2017. Fall detection in individuals with lower limb amputations using mobile phones: machine learning enhances robustness for real-world applications. JMIR mHealth uHealth 5 (10), e8201. https://doi.org/10.2196/mhealth.8201.
Shojaedini, Seyed Vahab, Beirami, Mohamad Javad, 2020. Mobile sensor based human activity recognition: distinguishing of challenging activities by applying long short-term memory deep learning modified by residual network concept. Biomed. Eng. Lett. 10 (3), 419–430. https://doi.org/10.1007/s13534-020-00160-x.
Tan, Mingxing, Le, Quoc, 2019. Efficient net: rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning. PMLR, pp. 6105–6114. https://doi.org/10.48550/arXiv.1905.11946.
Tao, Hai, Al-Sulttani, Ali Omran, Ameen, Ameen Mohammed Salih, Ali, Zainab Hasan, Al-Ansari, Nadhir, Salih, Sinan Q., Mostafa, Reham R., 2020. Training and testing data division influence on hybrid machine learning model process: application of

river flow forecasting. Complexity (1), 8844367. https://doi.org/10.1155/2020/8844367, 2020.
Chitty-Venkata, Teja, Krishna, Somani, Arun K., 2022. Neural architecture search survey: a hardware perspective. ACM Comput. Surv. 55 (4), 1–36. https://doi.org/10.1145/3524500.
Terroso, Miguel, Rosa, Natacha, Marques, Antonio Torres, Simoes, Ricardo, 2014. Physical consequences of falls in the elderly: a literature review from 1995 to 2010. Eur. Rev. Aging Phys. Activ. 11, 51–59. https://doi.org/10.1007/s11556-013-0134-8.
Tran, Cuong, Fioretto, Ferdinando, Kim, Jung-Eun, Naidu, Rakshit, 2022. Pruning has a disparate impact on model accuracy. Adv. Neural Inf. Process. Syst. 35, 17652–17664. https://doi.org/10.48550/arXiv.2205.13574.
Truong, Anh, Walters, Austin, Goodsitt, Jeremy, Hines, Keegan, Bayan Bruss, C., Farivar, Reza, 2019. Towards automated machine learning: evaluation and comparison of AutoML approaches and tools. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, pp. 1471–1479. https://doi.org/10.1109/ICTAI.2019.00209.
Van Kuppevelt, D., Meijer, Christiaan, Huber, Florian, Van Der Ploeg, A., Georgievska, S., Vincent, T. van Hees, 2020. Mcfly: automated deep learning on time series. SoftwareX 12, 100548. https://doi.org/10.1016/j.softx.2020.100548.
Vehtari, Aki, Gelman, Andrew, Gabry, Jonah, 2017. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Stat. Comput. 27, 1413–1432. https://doi.org/10.1007/s11222-016-9696-4.
Wan, Shaohua, Qi, Lianyong, Xu, Xiaolong, Tong, Chao, Gu, Zonghua, 2020. Deep learning models for real-time human activity recognition with smartphones. Mobile Network. Appl. 25 (2), 743–755. https://doi.org/10.1007/s11036-019-01445-x.
Wang, Xueyi, Ellul, Joshua, George, Azzopardi, 2020. Elderly fall detection systems: a literature survey. Front. Robot. AI 7, 71. https://doi.org/10.3389/frobt.2020.00071.
White, Colin, Jain, Paarth, Nayak, Sibasis, Ramakrishnan, Ganesh, 2022. Speeding up NAS with adaptive subset selection. arXiv preprint arXiv:2211.01454. https://doi.org/10.48550/arXiv.2211.01454.
Zebiah, S. Sherin, Vetha Shalomy, Ancy, Kachhap, Jyotsana, Tete, Nikita, Nancy, R., Ananthi, A., Prasanna, J., Subathra, M.S.P., Thomas George, S., 2023. Human fall detection using machine learning and deep learning techniques: a survey. In: 2023 4th International Conference on Signal Processing and Communication (ICSPC). IEEE, pp. 253–257. https://doi.org/10.1109/ICSPC57692.2023.10125648.
Zhu, Michael, Gupta, Suyog, 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. https://doi.org/10.48550/arXiv.1710.01878.
Zöller, Marc-André, Huber, Marco F., 2021. Benchmark and survey of automated machine learning frameworks. J. Artif. Intell. Res. 70, 409–472. https://doi.org/10.1613/jair.1.11854.