electronics

MDPI

Article

Low-Power On-Chip Implementation of Enhanced SVM Algorithm for Sensors Fusion-Based Activity Classification in Lightweighted Edge Devices

Juneseo Chang [1], Myeongjin Kang [2] and Daejin Park [3,*]

[1] Department of Computer Science and Engineering, Seoul National University, Seoul 08826, Korea; jschang0215@snu.ac.kr
[2] School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, Korea; audwls3158@knu.ac.kr
[3] School of Electronics Engineering, Kyungpook National University, Daegu 41566, Korea
* Correspondence: boltanut@knu.ac.kr; Tel.: +82-53-950-5548

Abstract: Smart homes assist users by providing convenient services from activity classification with the help of machine learning (ML) technology. However, most of the conventional high-performance ML algorithms require relatively high power consumption and memory usage due to their complex structure. Moreover, previous studies on lightweight ML/DL models for human activity classification still require relatively high resources for extremely resource-limited embedded systems; thus, they are inapplicable for smart homes' embedded system environments. Therefore, in this study, we propose a low-power, memory-efficient, high-speed ML algorithm for smart home activity data classification suitable for an extremely resource-constrained environment. We propose a method for comprehending smart home activity data as image data, hence using the MNIST dataset as a substitute for real-world activity data. The proposed ML algorithm consists of three parts: data preprocessing, training, and classification. In data preprocessing, training data of the same label are grouped into further detailed clusters. The training process generates hyperplanes by accumulating and thresholding from each cluster of preprocessed data. Finally, the classification process classifies input data by calculating the similarity between the input data and each hyperplane using the bitwise-operation-based error function. We verified our algorithm on 'Raspberry Pi 3' and 'STM32 Discovery board' embedded systems by loading trained hyperplanes and performing classification on 1000 training data. Compared to a linear support vector machine implemented from Tensorflow Lite, the proposed algorithm improved memory usage to 15.41%, power consumption to 41.7%, performance up to 50.4%, and power per accuracy to 39.2%. Moreover, compared to a convolutional neural network model, the proposed model improved memory usage to 15.41%, power consumption to 61.17%, performance to 57.6%, and power per accuracy to 55.4%.

Keywords: activity monitoring; machine learning; sensor fusion; SVM; edge-AI computing; energy-accuracy trade-off

Citation: Chang, J.; Kang, M.; Park, D. Low-Power On-Chip Implementation of Enhanced SVM Algorithm for Sensors Fusion-Based Activity Classification in Lightweighted Edge Devices. *Electronics* **2022**, *11*, 139. https://doi.org/10.3390/electronics11010139

Academic Editor: George A. Tsihrintzis

Received: 30 October 2021
Accepted: 31 December 2021
Published: 3 January 2022

1. Introduction

Today, platforms that provide convenient services using machine learning (ML) methods are rapidly developing in various fields. Among them, smart devices that provide appropriate feedback based on received signals are gaining popularity [1]. Various research projects are being conducted to classify human behavior with signals obtained from these devices. For instance, there have been studies on improving users' quality of life by classifying user activity data using signals obtained from wearable devices such as electrocardiography (ECG) [2,3], global positioning systems (GPS), and accelerometers [4].

Meanwhile, utilizing data collected from various sensors enables a more complex understanding of the situation. Sensor fusion reduces software complexity by hiding

physical sensor layers and offers organized, fine quality input data for applications [5]. Therefore, many studies are interested in finding an efficient method for fusing and utilizing various sensor data. For instance, to determine the state of numerous edge devices, a study used a QR code generated from power consumption data of edge devices. By handling complex data as efficient image data, they classified error states with reduced load on edge devices [6].

Because smart devices are connected to other devices via wireless protocols, it is possible to provide complex services using various sensor signals [7]. Therefore, based on Internet of Things (IoT) sensors and ML technologies, smart homes that monitor the house condition and automatically adjust appliances were made possible [8]. Numerous studies aim to solve social problems by classifying human behavior from information that was gathered from many sensors in smart homes. For instance, the smart home is being discussed as a solution that can socially and medically assist the infirm by monitoring them through numerous smart devices [9]. Furthermore, various research is being conducted on how smart home data can be processed and analyzed efficiently [10].

In general, the architecture of many smart home models consists of low-power embedded processors due to their energy consumption [11]. Therefore, the performance of software installed in smart homes should be optimized to achieve the utmost performance from limited resources. To enhance the performance in low-power environments, various software and hardware-accelerated optimization techniques are being developed [12,13]. Memory usage is also a critical issue in low-power embedded environments. Therefore, many studies are also focusing on developing algorithms to reduce memory usage as a solution to the limited memory of the low-power embedded processors used in smart home models [14].

ML has made great advancements in analyzing data collected from a smart home [15]. However, most high-performance ML algorithms were unsuitable for use on edge due to their size and power consumption; thus, in recent years, there have been significant efforts to develop ML algorithms and systems for edge devices [16]. As a result, a low-power, memory-efficient ML algorithm optimized for smart home data should be designed in order to implement ML in the smart home model efficiently.

2. Related Works

Human activity recognition (HAR) is crucial due to its ability to learn high-level human activity information from raw sensor data. The HAR problem is equivalent to a pattern recognition (PR) problem [17]. The PR problem is solved in the order of activity signal, feature extraction, model training, and activity information, and many studies have been conducted at each stage. The previous works can be broadly classified into studies focusing on sensor type and feature extraction/training.

Chavarriaga [18] classified sensor modalities into body-worn, object, and ambient sensors. Body-worn sensors, which are the most common modality for HAR, are commonly used based on deep learning [17]. Object sensors are frequently attached to objects to detect their movement [18]. Ambient sensors such as radar, sound, and temperature sensors are mostly embedded in the user environment, and they detect signals from humans interacting with the environment. Several papers have used ambient sensors to detect daily activities [19]. Hybrid sensors are a combination of different types of sensors for HAR. However, there are only a few works that combined various sensors for more accurate HAR [20].

In HAR, traditional PR tactics have achieved remarkable progress [21]. Recently, however, deep learning research that integrates feature extraction and training processes have become mainstream. Deep models can be largely divided into Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Stacked Autoencoders (SAE), Recurrent neural networks (RNN), hybrid models, etc.

For studies that applied DNN for HAR, after extracting hand-engineered features from the sensors, those features are fed into a DNN model [22]. However, because the feature extraction is done manually, it may be difficult to utilize the model in general cases.

While using CNN to HAR, there are several factors to consider: input adaptation, pooling, and weight-sharing. In contrast to images, most HAR sensor data are time-series data readings. Therefore, we should transform input data, and there are two main types: model-driven and data-driven. In the data-driven approach, 1D convolution is applied to each dimension, where the dimensions are used as channels [23]. In a model-driven approach, the inputs are resized to a virtual 2D image in order to use 2D convolution. This is frequently used in conjunction with non-trivial input tweaking approaches [24]. CNN commonly uses the convolution-pooling combination [24]. Moreover, pooling can speed up the training process on big data sets and reduce overfitting [25]. Weight sharing [26] is a useful technique for improving the speed of the training. According to a study [27], partial weight-sharing might enhance CNN's performance.

Stacked autoencoder (SAE) has the benefit of unsupervised feature extraction for HAR. However, SAE is very reliant on its layers and activation functions, making it difficult to find the optimal solutions [28].

Many studies on solving HAR problems by RNN models achieved good performance in resource-constrained environments [29].

Hybrid models, such as the combination of CNN and RNN, are currently gaining popularity. There are several studies on combining CNN and RNN for HAR [30].

There are also many studies on the lightweight deep learning model. Preeti and Mansaf [31] developed a model that even runs even on Raspberry 3Pi by optimizing parameters by combining RNN and LSTM.

3. Problem Statement

3.1. Smart Home Model

In this work, we assumed a typical smart home architecture as illustrated in Figure 1. The smart home is equipped with event-driven IoT sensors which are directly connected to the main edge device. The main edge device is loaded with a pre-trained model used for human activity classification and is connected to a high-performance server [32].

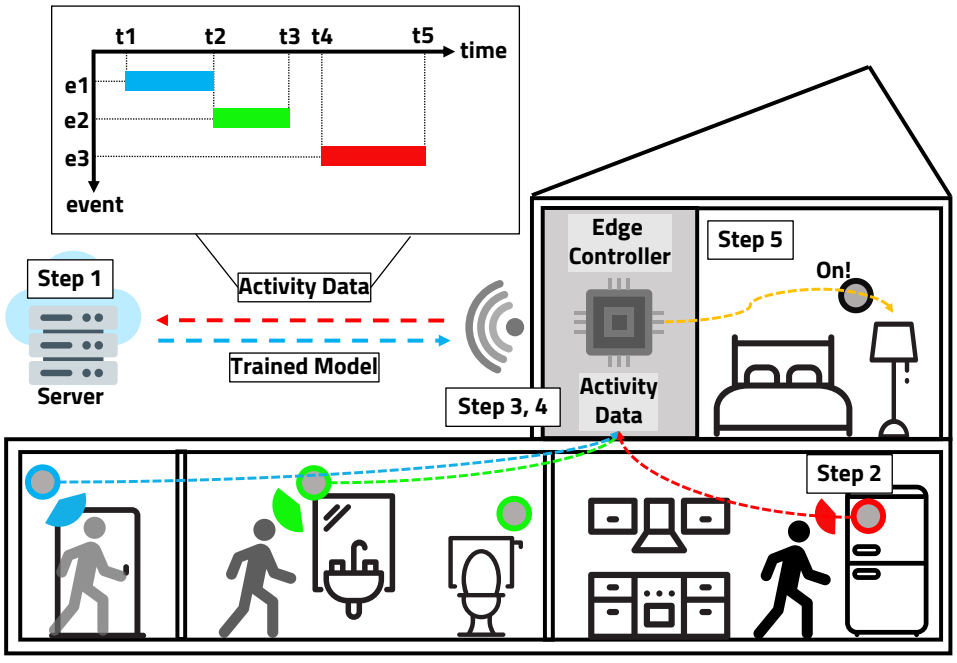

Figure 1. Overview of the smart home model.

In Figure 1, we also illustrated the overall operation method of our smart home model, which works according to the following steps. Step 1. The high-performance server trains the human activity classification model and sends the trained model to the main edge device. Step 2. IoT sensors detect human behaviors and send the detected signals to the main edge device. Step 3. The main edge device saves the data from various sensors in a buffer as activity data. Step 4. In the main edge device, the trained model classifies the activity data to human activity patterns. Step 5. The main edge device makes an appropriate response according to the classified activity pattern.

3.2. Activity Data

Converting and using time series data to image form is a widely used technique. When we plot the signals obtained from the multiple sensors according to time, it will appear as the sensor activity data as shown in Figure 2. We then convert the sensor activity data to a grayscale image in which its row is sensors and column is time to abstract the data more efficiently. Each pixel value in the grayscale image is the sensor's signal level in the corresponding time and sensor. Therefore, by handling grayscale images, we can process sensor activity data more conveniently.

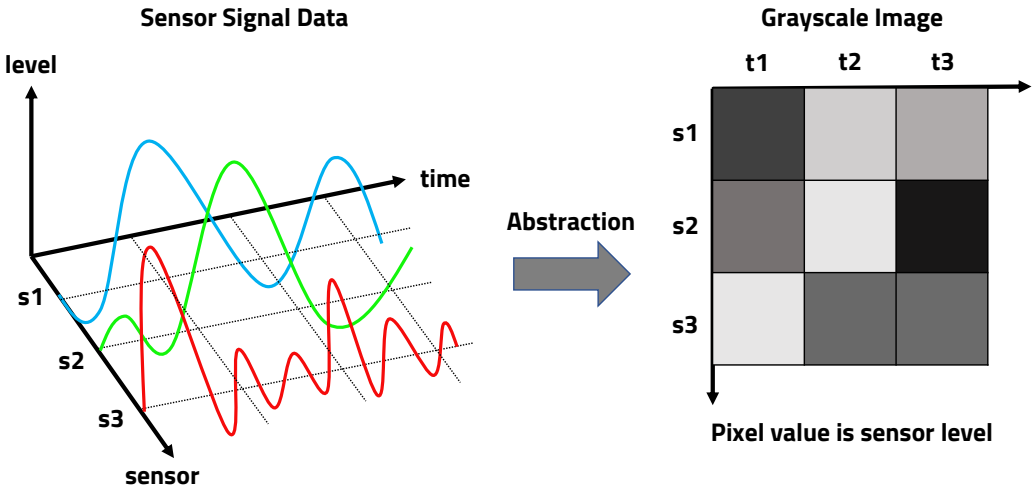

Figure 2. Conversion of sensor signal data to a grayscale image.

Although handling a grayscale image is convenient, converting it to a binarized image is more efficient, especially in low-power, limited-resource applications. To generate a binarized image, we apply a threshold plane to the sensor signal data as shown in Figure 3. The corresponding pixel in the binarized image is set to 1 if the sensor's signal level is above the threshold or 0 otherwise. As a result, by converting sensor signal data to a binarized image using a threshold plane, we can reduce data size while minimizing data loss.

Figure 4 is the detailed representation of the sensors' signals data as a binarized image. In this paper, for activity data, we use the converted binary image. To handle time data, we manifest the events' occurrence by a time range [13]. The activity data AD is defined as the following Equation (1).

$$AD(s,\ r_i) = \begin{cases} 1 & \text{if } \exists t \in r_i \textit{ such that } f(s,t) = 1 \\ 0 & \text{otherwise} \end{cases} \quad s \in S,\ r_i \in TR \tag{1}$$

where:
S = Sensors in smart home
$f(s,\ t)$ = Activation of sensor s in time t (0 not activated, 1 activated)
Δt = Constant time interval between activity data
$TR = \{(n\Delta t\ :\ (n+1)\Delta t) \mid n \in \mathbb{N}\}$ Set of time ranges

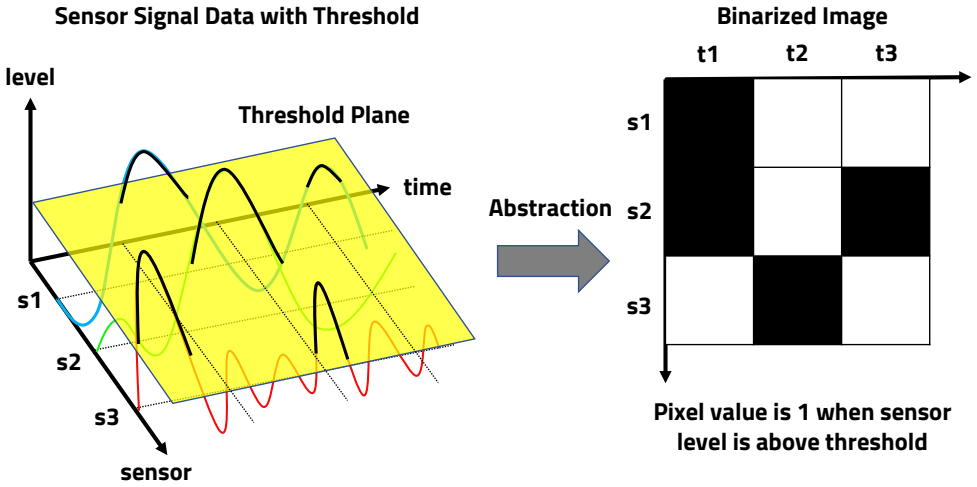

Figure 3. Conversion of sensor signal data to a binarized image by threshold.

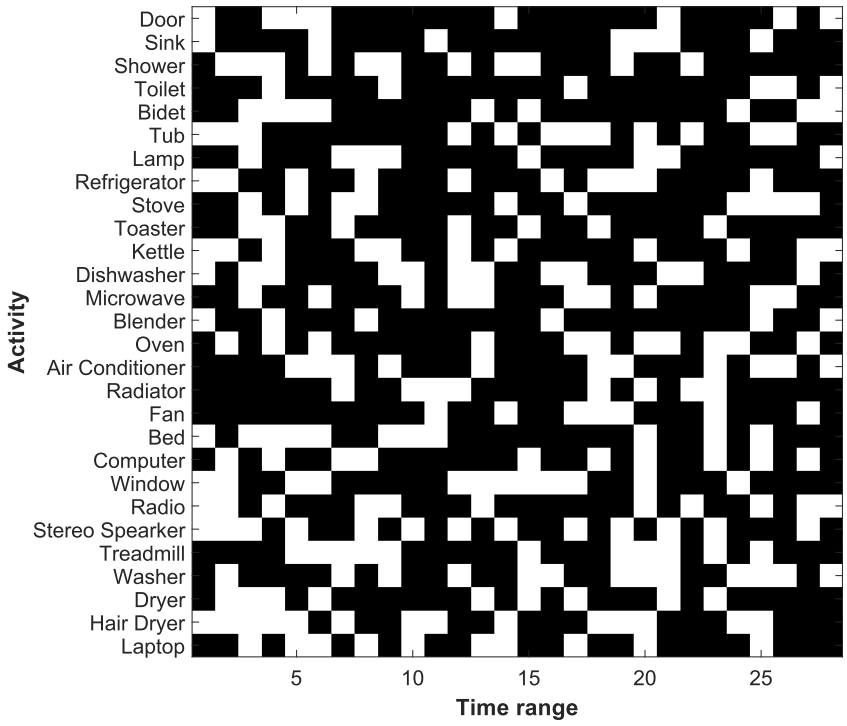

Figure 4. Visualization of activity data.

Each entry in the activity data represents the activation of the sensor in the corresponding time interval. Such binarized images can easily perform high-speed operations such as boolean products and can be efficiently stored and accessed from memory. Because the activity data can be comprehended as image data as shown in Figure 4, the classification of activity data is equivalent to image classification.

We utilize the numeric handwriting image dataset MNIST [33] as the activity data, considering the characteristics of activity data as an image. Since it was impossible to utilize real-world activity data as our dataset due to experimental constraints, and the MNIST dataset and activity data are very similar when converting both data to image form, we used the MNIST dataset as a substitute for activity data. Figure 5 shows how the activity data is similar to the MNIST image. If we transform activity data into an image, it is similar to the MNIST dataset because it has a constant pattern for a particular label. Therefore, if we binarize the MNIST image to indicate sensor activation, it will take the form of activity data that this study assumes.

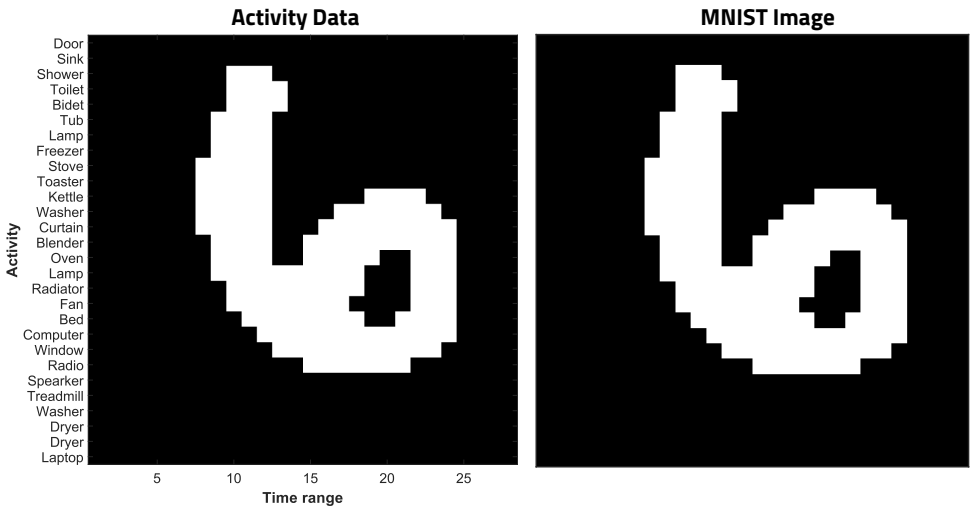

Figure 5. Similarity between activity data and MNIST image.

However, it is practically impossible to collect as much activity data as the MNIST dataset because the activity data is challenging to collect and utilize due to privacy issues [34]. In other words, only a small amount of activity data is available compared to the MNIST dataset. Therefore, we use 10% of the MNIST dataset chosen randomly as activity data.

3.3. Research Objectives

Although there exist some studies on lightweight activity data classification models, they either require relatively high resources for resource-constrained embedded systems, or detailed resource measurements such as power consumption were not evaluated on embedded boards. Additionally, deploying an activity classification model on a resource-constrained embedded system can bring economic benefits for businesses. Therefore, in this study, we propose an efficient ML algorithm that can operate in smart homes composed of resource-limited embedded systems. In order to meet the embedded system environment, our algorithm should satisfy the following conditions:

- Low power consumption
- Small memory usage
- Fast run time

We chose the linear support vector machine (LSVM) [35] as a baseline project and aimed to study a lighter, more power-efficient algorithm. LSVM is a widely used ML algorithm in embedded environments due to its small memory usage, and computation [36]. Because LSVM generates hyperplanes without semantics, we focused on the hyperplane generation technique that considers the activity data characteristics. Therefore, we aimed to develop an algorithm with sufficient accuracy with fewer hyperplanes than the established ML algorithms. We used the MNIST dataset as a substitution for smart home activity data to evaluate the model accuracy and performance on an embedded board, thus verifying the model's suitability in real-world smart homes.

4. Proposed Method

In this work, we propose a high-speed, memory-efficient data preprocessing, training, and classification method for our smart home model. Figure 6 shows the overall structure of the proposed algorithms in this study. First, we preprocess the training data by clustering it into similar groups. Then, the training algorithm generates memory-efficient hyperplanes from the preprocessed training dataset. In our work, the 'hyperplane' refers to the plane on which the model makes decisions based on calculating the distance between the input

data and the plane. Finally, the classification process classifies the activity data in an embedded environment.

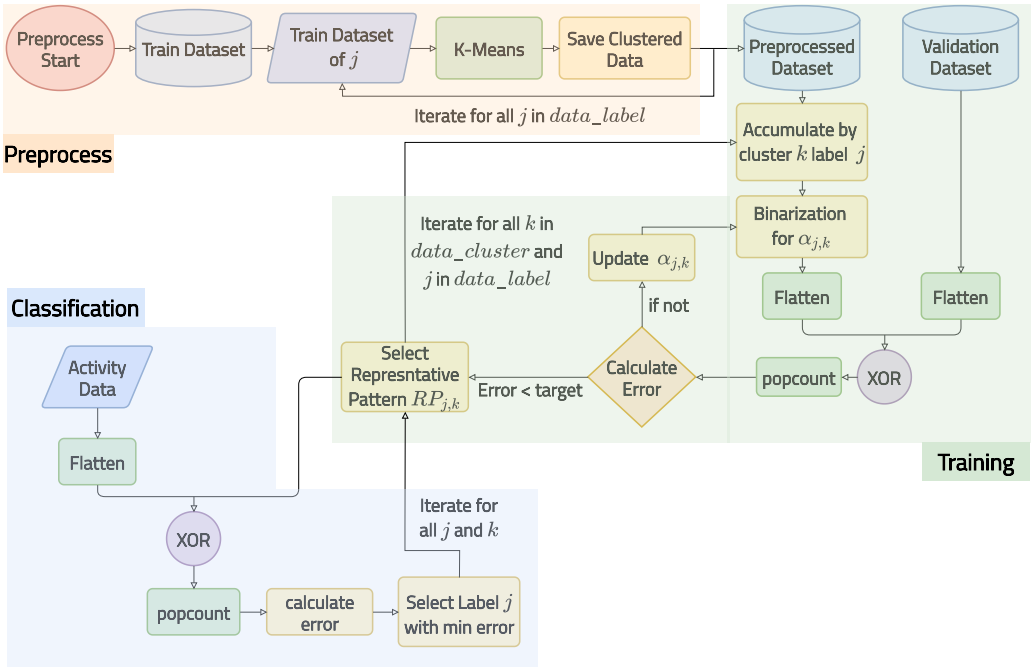

Figure 6. Overall strucutre of the proposed algorithm.

4.1. Data Preprocessing

The data preprocessing step is important for generating efficient hyperplanes. In this study, we summed up and binarized the training dataset to generate hyperplanes (which will be discussed further in the next section). Simply accumulating and binarizing the data can reduce data size while maintaining the data's overall characteristics; however, the image loses the data's detailed characteristics. We observed that the detailed characteristics had a significant impact on classification. We also observed that the images in the same label of the MNIST dataset had slightly different traits. For example, some handwriting was in italic, while some were in bold format. To solve this problem, we performed clustering to the training data of the same label to make our hyperplanes represent the data's detailed features.

Figure 7 shows the proposed method for the preprocessing data. Algorithm 1 shows the pseudo-code of the process. 'Preprocess' is a function that returns clustered data from training data for each label. To group data from the same label, we first grouped training data according to their label. Then, we applied k-means clustering [37] to the data that were grouped for each label. Finally, data belonging to the same cluster of the same label were stored as preprocessed data.

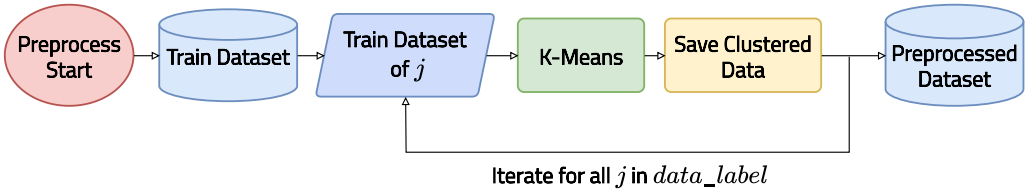

Figure 7. Overall algorithm of the preprocess.

Algorithm 1: Pseudo code of data preprocess

```
data_label: labels of dataset
KMeans(cluster_num): k-means clustering that forms cluster_num clusters
preds: array of clustering result for each entry of input
Input: train_dataset, cluster_num
Output: preprocessed_dataset
Function Preprocess(train_dataset, cluster_num):
    foreach img ∈ train_dataset do
        img_label = img.data_label
        grouped_image[img_label].append(img)
    foreach j ∈ data_label do
        model = KMeans(cluster_num)
        model.fit(grouped_image[j])
        preds = model.labels
        foreach img ∈ grouped_imagejl] do
            cluster = preds[img.index]
            preprocessed_dataset[j][cluster] = img
```

We implemented the clustering process using the k-means function of scikit-learn [38] with 5 cluster numbers. The equation for k-means clustering for data of label j is shown in (2).

$$argmin_{X_j} \sum_{i=1}^{n} \sum_{t \in T_j} \|t - \mu_i\|^2 \tag{2}$$

where:
n = Number of clusters
T_j = Training dataset of label j
$X_j = \{X_{j,1}, X_{j,2}, ..., X_{j,k}\}$; $X_{j,i}$ is i-th clustered dataset for label j
μ_i = Mean point in $X_{j,i}$

Figure 8 shows the results of clustering. The data in the same row represents samples of data classified into the same cluster. We can see that k-means clustering clearly distinguishes data with slightly different characteristics from data within the same label.

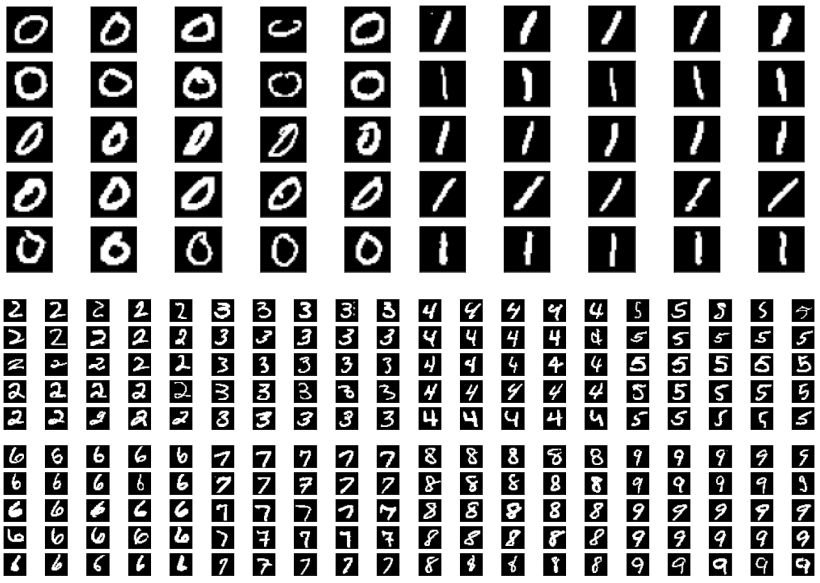

Figure 8. Clustering result.

4.2. Training Algorithm for Hyperplane Generation

We accumulated and binarized the preprocessed data and reshaped them to generate representative patterns which were equivalent to hyperplanes. Figure 9 shows the proposed training algorithm for generating hyperplanes. Algorithm 2 is the pseudo-code of the training process. 'Train' is a function that returns the representative patterns for each cluster in each label. The representative patterns are generated from the preprocessed dataset and the validation dataset evaluates them.

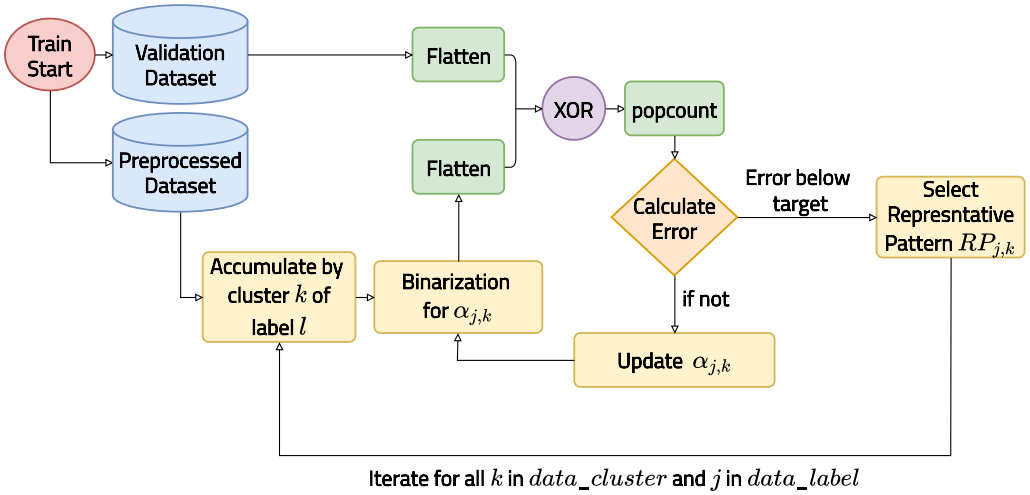

Figure 9. Overall algorithm of the training process.

Algorithm 2: Pseudo code of training algorithm

```
preprocessed_dataset[j][k]: preprocessed data of cluster k of label j
validation_dataset[j][k]: validation dataset is split into cluster k of label j
flatten(img): return img collapsed into one dimension
representative_pattern[j][k]: representative pattern for cluster k of label j
update(val, error): update val according to error
popcount(n): return number of set bits in n
Input: validation_dataset, preprocessed_dataset
Output: representative_pattern
Function Train(validation_dataset, preprocessed_dataset):
    while error > target_error do
        foreach j ∈ data_label and k ∈ data_cluster do
            foreach x ∈ preprocessed_dataset[j][k] do
                R[j][k] = R[j][k] + x
            foreach pixel ∈ R[j][k] do
                if pixel > alpha[j][k] then
                    pixel = 1
                else
                    pixel = 0
            RP[j][k] ⟵ flatten(R[j][k])
            foreach td ∈ validation_dataset[j][k] do
                td ⟵ flatten(td)
                error+ = popcount(RP[j][k] ⊕ td)
            update(alpha[j][k], error)
```

To generate representative pattern $RP_{j,k}$ for each cluster k of label j , we accumulated the preprocessed dataset to an image $R_{j,k}$. The equation for accumulating preprocessed dataset is shown in Equation (3).

$$R_{j,k}(x, y) = \sum_{x_{j,k} \in X_{j,k}} x_{j,k}(x, y) \tag{3}$$

where:
$R_{j,k}$ = Accumulated image of preprocessed dataset for cluster k of label j
$X_{j,k}$ = Preprocessed dataset for cluster k of label j
(x, y) = Pixel position in image

Simply summing up the preprocessed data produces $R_{j,k}$ with the same importance for both the inliers and the outliers. Therefore, we binarized pixel values according to the threshold $\alpha_{j,k}$ to reduce the outliers' effect. As a result, we could get $R_{j,k}$ which represents the inlier better. The equation for binarizing $R_{j,k}$ is expressed in Equation (4).

$$R_{j,k}(x, y) = \begin{cases} 1 & \text{if } R_{j,k}(x, y) > \alpha_{j,k} \\ 0 & \text{otherwise} \end{cases} \quad \alpha_{j,k} \text{is a trainable parameter} \tag{4}$$

We flattened the $R_{j,k}$ to the representative pattern $RP_{j,k}$. At the same time, we split the validation datset of label j to k batches, and applied XOR operation between $RP_{j,k}$ and $td_{j,k}$, where $td_{j,k}$ is an flattened element of validation dataset for cluster k of label l. From the XOR value, we determined the error between $RP_{j,k}$ and $td_{j,k}$ by counting the number of the set value of the XOR value. The equation for error caluclation is shown in Equation (5).

$$error = \sum_{td_{j,k} \in TD_j} popcount(RP_{j,k} \oplus td_{j,k}) \tag{5}$$

where:
$RP_{j,k} = vec(R_{j,k})$
$TD_{j,k} = \{x_{j,k} \mid x_{j,k} = vec(y_{j,k}),\ y_{j,k} \in Y_{j,k}\}$
$Y_{j,k}$ = Validation dataset for batch k of label j
$popcount(n)$ = Number of set bits in n

Based on the error, we either updated $\alpha_{j,k}$ or finished the training process when the error was below the target. We determined $\alpha_{j,k}$ by creating a lookup table for $\alpha_{j,k}$ and its error. Figure 10 shows the graph between epoch and mean error in the training and validation dataset, where the mean error is the mean value of the error defined in Equation (5) of each dataset per epoch. Figure 11 visualizes the hyperplanes we generated. However, due to the complexity of defining the space of the hyperplane mathematically, we left the part of the defining hyperplane as a mathematical equation. As can be seen in the figure, the hyperplanes soundly reflect the characteristics of each label's cluster.

4.3. Activity Data Classification at Edge

Because the classification process is performed on a low-power edge, it must be able to function well even with a small amount of resources. In this study, we propose a high-speed classification algorithm based on bitwise operations, suitable for our edge environment. Figure 12 shows the proposed classification algorithm. Algorithm 3 shows the pseudo-code of the process. 'Classify' is a function that determines the input activity data's label. The classification process is similar to the error calculation in the training algorithm. First, we calculated the error by applying the XOR and popcount operation between the representative plane $RP_{j,k}$ and the input activity data AD. On low-power processors, bitwise operations perform faster than multiplication and addition. Therefore,

we could reduce the overhead and improve our algorithm's speed. We then selected the label of minimum error as shown in Equation (6).

$$\hat{j} = argmin_j(error(RP_{j,k},\ AD)) \tag{6}$$

where:
$error(RP_{j,k}, AD) = popcount(RP_{j,k} \oplus AD)$
AD = Input activity data

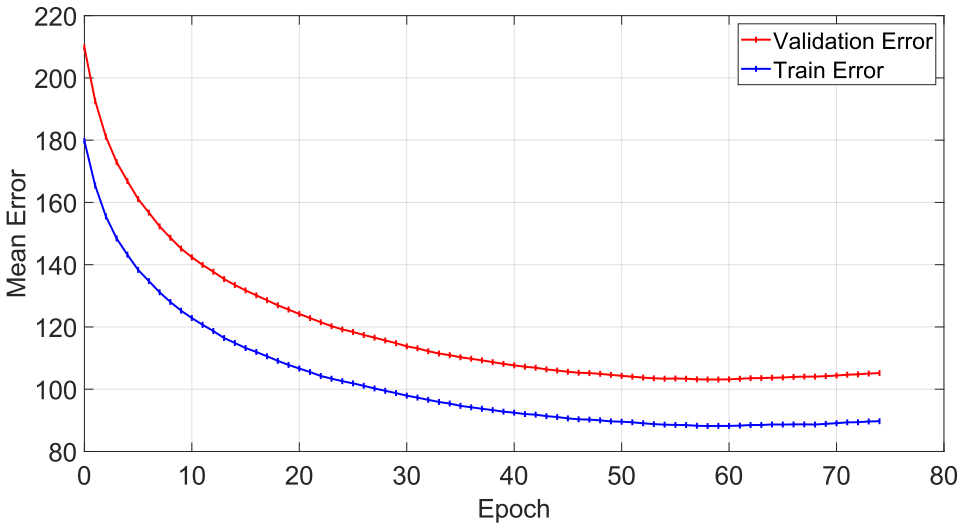

Figure 10. Mean error of training and validation data over epochs.

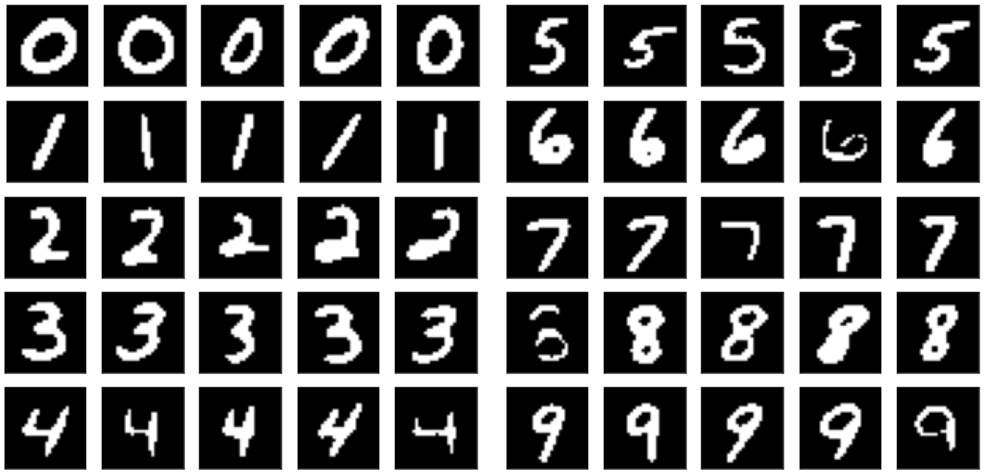

Figure 11. Visualization of hyperplanes.

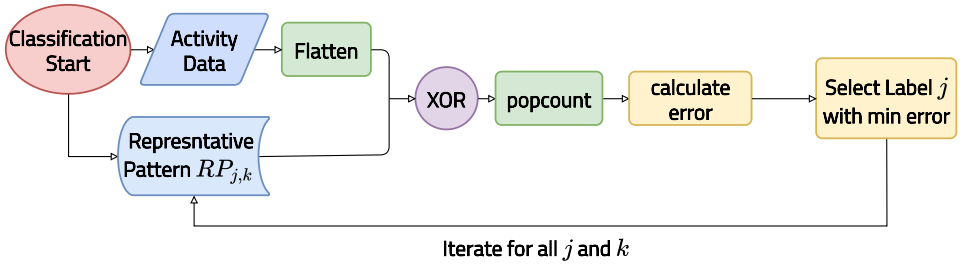

Figure 12. Overall algorithm of the classification process.

Algorithm 3: Pseudo code of classification algorithm

Input: *activity_data*
Output: *determined_label*
Function `Classify(`*activity_data*`)`**:**
  $activity\_data \longleftarrow flatten(activity\_data)$
  $min \longleftarrow \infty$
  foreach $label \in data\_label$ *and* $cluster \in data\_cluster$ **do**
    $error[label][cluster] = popcount(representative\_img[label][cluster] \oplus activity\_data)$
    if $error[label][cluster] < min$ **then**
      $min \longleftarrow error[label][cluster]$
      $determined\_label = label$

5. Experiment & Measurement Results

The proposed model in this paper assumes execution at the edge through the high-speed, low-power classification process. In this section, we verified whether the proposed algorithm is suitable for application on edge-embedded systems with the experimental setup 'Raspberry Pi 3' and 'STM32 Discovery board' as shown in Figure 13.

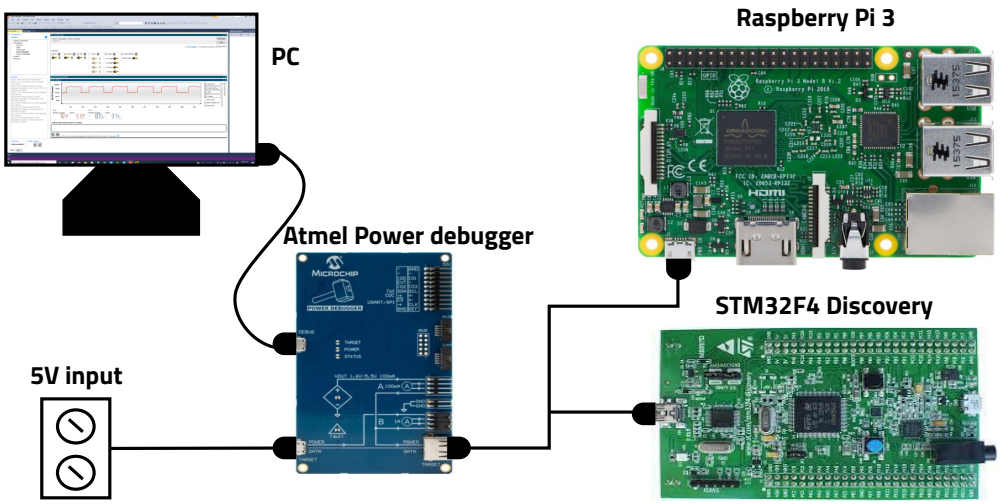

Figure 13. Experimental setup.

The classification edge was implemented at C level to compare with the baseline project LSVM. The LSVM model was trained using Tensorflow Lite [39], and was implemented at C level as well. We also compared the proposed model with the CNN model, which is trained using Tensorflow Lite and implemented at C level. The CNN model is composed of two layers: the first layer is of a four channel convolution layer with $3 * 3$ kernel size and ReLU activation function with max-pooling; the second layer is of eight channels convolution layer with $3 * 3$ kernel size and ReLU activation function with max-pooling. The dropout is set to 0.5 and the total number of parameters was 2346. The number of clusters was set to five since our model had reasonable accuracy, memory usage, and power consumption under that cluster number. For a more accurate measurement, algorithm performance and accuracy were measured using 'Raspberry Pi 3', and memory usage and power consumption were measured on 'STM32 Discovery board' and 'Atmel Power Dubugger' to ensure operation in more limited embedded systems. We repeated the classification process of the proposed model, LSVM model, and the CNN model with a short time interval between each classification process for 1000 samples of test data.

5.1. Performance

Since the classification process is executed repeatedly in a short-time period on real-world smart homes, it needs to be operated in real-time in a resource-limited environment. As illustrated in Figure 14a, the proposed model took 44 µs on average to perform all the test benches on the 'Raspberry Pi 3' board, while the LSVM model took 82 µs, and the CNN model took 101 µs, resulting in a 46.3% and 57.6% reduction in execution time. Also, the proposed model took 50 µs on the 'STM32 Discovery board' board, while the LSVM model took 92 µs, and the CNN model took 118 µs, resulting in a 45.6% and 56.4% reduction in execution time. This is the result of the proposed model reducing overheads by computing error with the bitwise operation. Moreover, when limiting the memory usage to simulate operation in a smaller embedded system, the difference in execution time between the two models becomes up to 50.4% for the LSVM model and 55.6% for the CNN model on the 'STM32 Discovery board', as shown in Figure 14b. This result is done without any CPU or memory acceleration, and therefore we believe that the proposed algorithm can have a greater effect when the edge is run on smaller hardware.

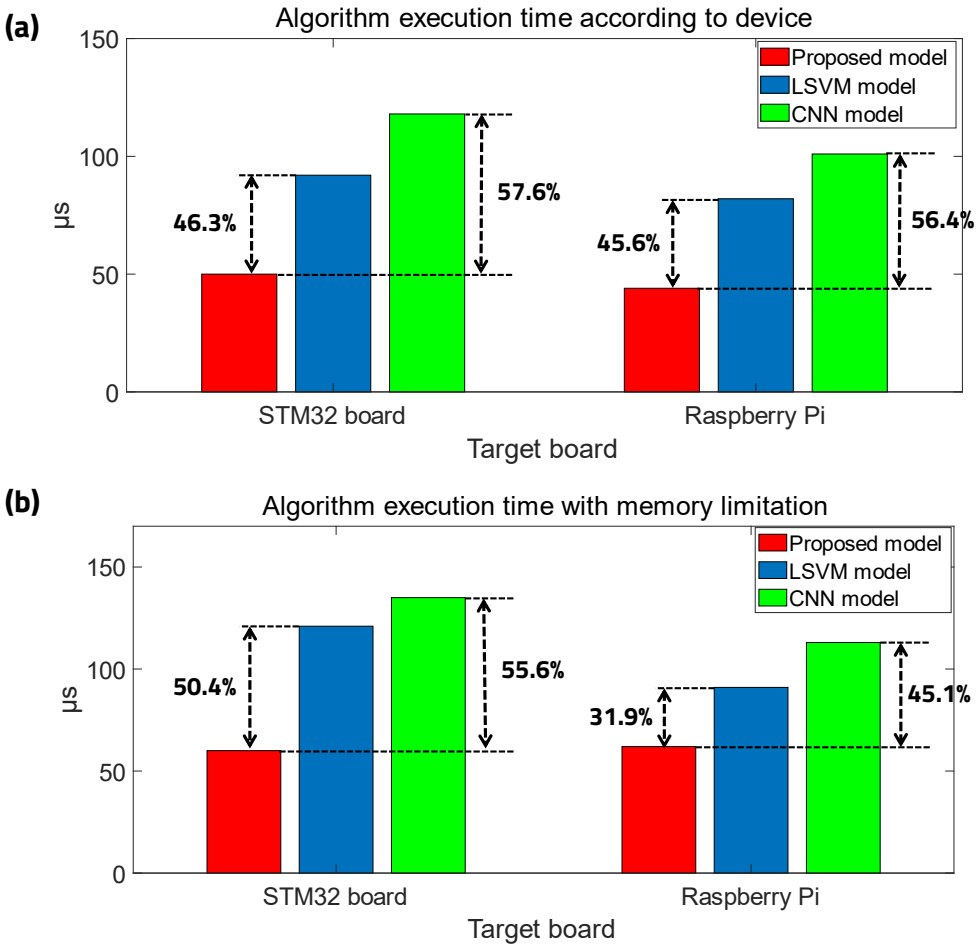

Figure 14. (**a**) Execution time of each algorithm. (**b**) Execution time of each algorithm with memory limitation.

5.2. Memory Usage

In an embedded system, memory usage is an important aspect to consider. In particular, the peak value in memory usage over time is critical because it determines the overall memory size used in the embedded system. Therefore, we tested the classification process's memory usage over the test dataset to compare memory usage over time between the proposed model and the LSVM and CNN model. We used the Valgrind Massif profiler

to measure heap and stack memory usage and repeated the classification of 1000 test data with 0.5 s intervals [40].

As shown in Figure 15, the proposed algorithm's peak memory usage was 81.94 KB, while the LSVM and CNN model used 96.39 KB at its peak. We believe that the peak memory usage of the LSVM and CNN model is of the same value because both models were implemented using Tensorflow Lite. Thus, the proposed model's memory usage had lower volatility over time and reduced peak memory usage by 15.41% than the LSVM and CNN model.

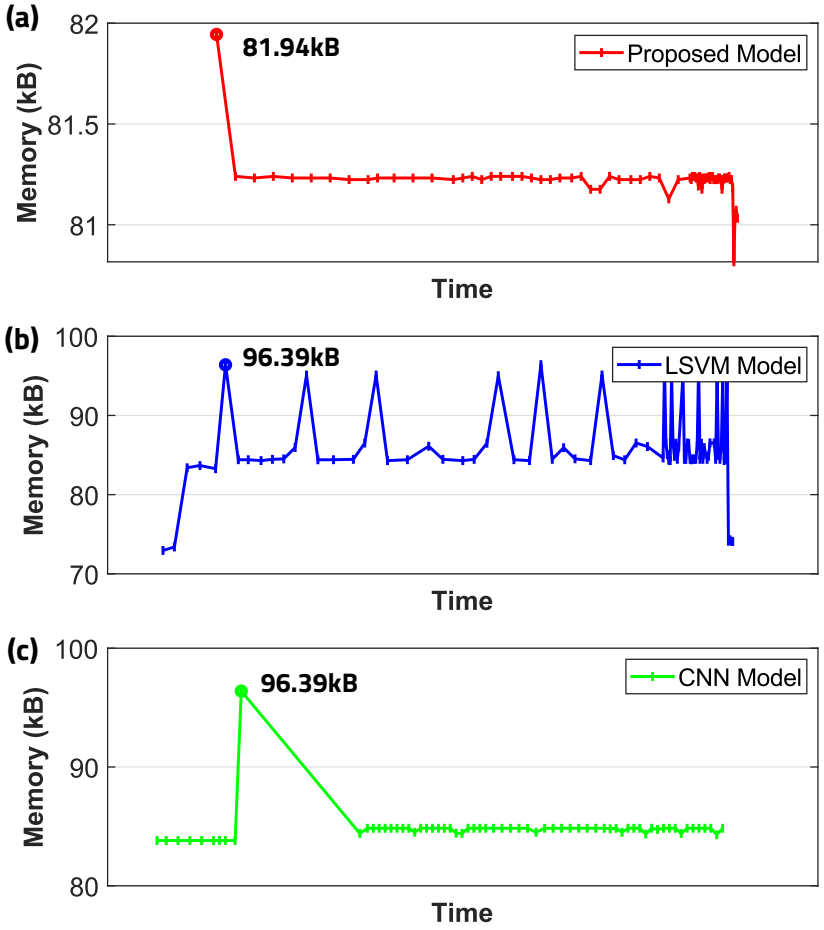

Figure 15. (**a**) Memory usage of the proposed model. (**b**) Memory usage of the LSVM model. (**c**) Memory usage of the CNN model.

5.3. Power Consumption

Power consumption is an important consideration when choosing an algorithm to run in an embedded system [3]. In this experiment, we used the 'STM32 Discovery' board to ensure operation in a smaller embedded system. 'Atmel Power Debugger' was used to measure the power consumed for the operation on this board [41]. To compare the proposed algorithm with the LSVM and CNN model, the classification operation, which compares 1000 test data samples with 10 hyperplanes, was repeated at regular time intervals. Because the 'STM32 Discovery board' takes too long to run the test bench, we reduced the test bench's size while keeping the total computation number the same for the proposed and the LSVM models. Figure 16 shows the power analysis of the proposed model, LSVM model, and CNN model on the 'STM32 Discovery board'. As shown in Figure 16, the standby current and active current of the models are the same. However, there is a difference in the average current due to each algorithm's performance. As shown in Section 5.1, the proposed model has a faster data processing speed compared to the LSVM and CNN

model. A faster data processing speed means a reduction in chip operating time, which leads to a reduction in the entire system's power consumption.

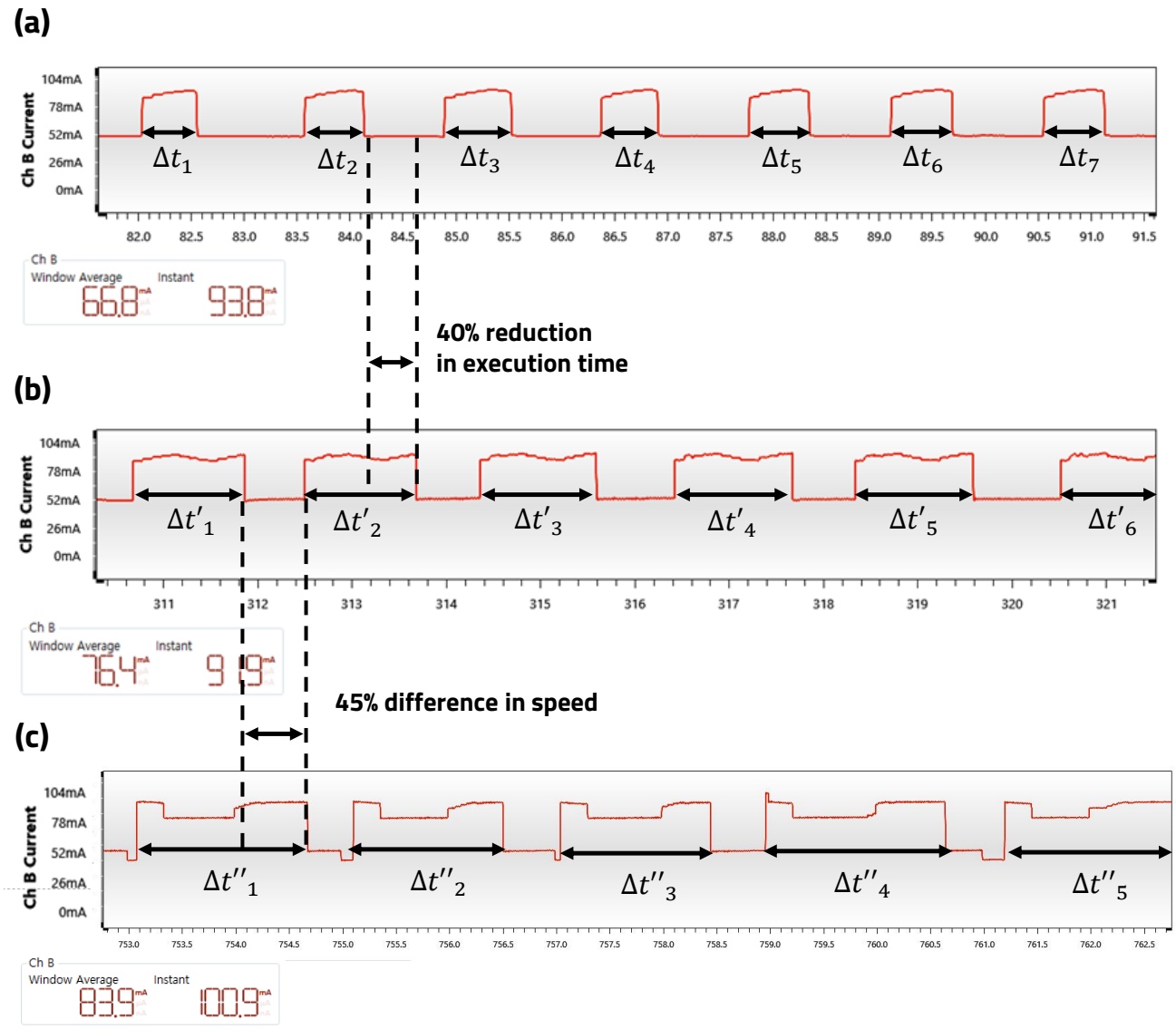

Figure 16. (**a**) Current data of the proposed model. (**b**) Current data of the LSVM model. (**c**) Current data of the CNN model.

Because a constant 5 V voltage drives the 'STM32 Discovery board', the total energy consumption is proportional to the amount of current used. The equations for comparing the energy consumption of the proposed model, LSVM model, and CNN model are as follows.

$$E_{proposed} = \int P_i(t) - P_{standby}\, dt = \sum_{n=1}^{k} (\Delta t) * P_{average} \tag{7}$$

$$E_{LSVM} = \int P_i(t) - P_{standby}\, dt = \sum_{n=1}^{k} (\Delta t') * P_{average} \tag{8}$$

$$E_{CNN} = \int P_i(t) - P_{standby}\, dt = \sum_{n=1}^{k} (\Delta t'') * P_{average} \tag{9}$$

The energy consumed in the classification can be calculated by the Equations (7)–(9), which is determined by the difference between Δt_n, $\Delta t'_n$, and $\Delta t''_n$.

$$(\Delta t_n) < (\Delta t'_n) < (\Delta t''_n) \tag{10}$$

Due to the proposed model's increased performance, as shown in Equation (10), Δt_n is shorter than $\Delta t'_n$ and $\Delta t''_n$ This shows that the increase in speed due to the bitwise operation of the proposed model leads to a decrease in the processor's energy consumption. Looking at the results measured using the actual 'STM32 Discovery board' the LSVM model uses about 510×10^{-6} [J] to perform test bench operation, the CNN model uses about 765×10^{-6} [J], and the proposed model consumes 297×10^{-6} [J] of energy. The proposed model's lower power consumption results from the shorter running time.

5.4. Model Evaluation Metrics

This section presents our model's evaluation metrics and compares them with the LSVM and CNN model. We evaluated our model based on 1000 test data samples. The confusion matrix for our model is shown in Figure 17. The proposed model's accuracy, precision, recall, and f1-score are listed in Table 1 and compared with the LSVM and CNN models.

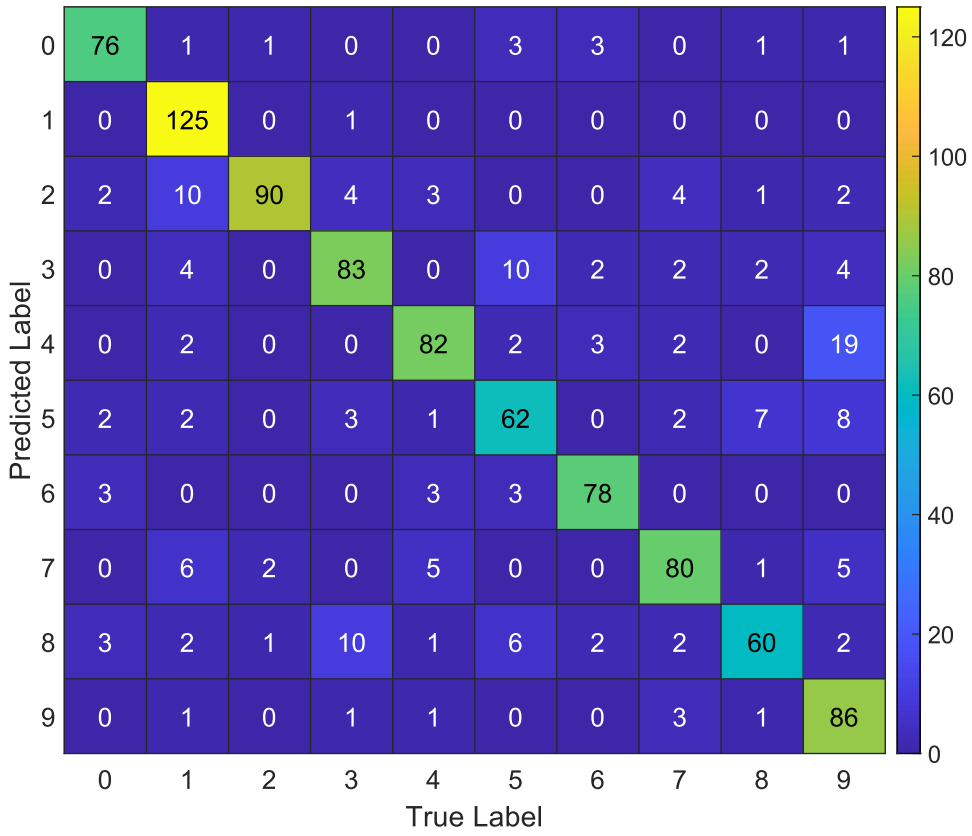

Figure 17. Confusion matrix of the proposed model.

Table 1. Evaluation metrics of the proposed, LSVM, CNN model.

	Accuracy	Precision	Recall	F1-Score
Proposed	82.2%	82.75%	81.89%	82.32%
LSVM	85.8%	86.19%	85.99%	86.09%
CNN	94.6%	94.69%	94.72%	94.71%

Since the proposed model has a smaller model size than the LSVM and CNN models, its slightly lower evaluation metrics are justifiable. Furthermore, considering the model's power consumption, we calculated the power consumption per evaluation metrics in Table 2. As shown in Table 2, the proposed model had much lower power consumption per evaluation metrics than the LSVM and CNN models.

Table 2. Power consumption per evaluation metric of the proposed, LSVM, CNN model. Lower value means more power efficient.

	Power per Accuracy	Power per Precision	Power per Recall	Power per F1-Score
Proposed	3.61 μJ/%	3.59 μJ/%	3.63 μJ/%	3.61 μJ/%
LSVM	5.94 μJ/%	5.92 μJ/%	5.93 μJ/%	5.92 μJ/%
CNN	8.09 μJ/%	8.08 μJ/%	8.08 μJ/%	8.08 μJ/%

6. Discussion

Our proposed model improved power consumption to 41.7% and memory usage to 15.41%, while the overall accuracy was only 3.6% lower than the LSVM model. Moreover, for the CNN model, our model improved power consumption to 61.17% and memory usage to 15.41%, while the accuracy was 12.4% lower. Since most of the existing human activity recognition models are based on traditional ML/DL such as SVM and CNN, we believe that our proposed model is very suitable for the smart home model. The activity classification is executed repeatedly in a resource-constrained device; thus, power consumption is an essential consideration. Additionally, the model's memory usage is directly connected with the cost of the device. Therefore, power consumption and memory usage are as crucial as model accuracy, and a slight loss in accuracy can be justified by the improvement in power and memory consumption.

Our former work dealt with a high-speed, memory-efficient ML algorithm by reducing the size of hyperplanes and utilizing a high-speed string comparison algorithm to measure similarity between input and each hyperplane [42]. However, in this study, we improved the accuracy to 5.62% and the running time to 75.02% by adding preprocessing and using bitwise-operation-based error calculations. Furthermore, we implemented the entire code in C language without external libraries and verified the algorithm's performance on an embedded board. Our improvement taught us how important preprocessing is in a lightweight ML algorithm. We believe that clustering data in preprocessing optimized the number and importance of the parameters, thereby resulting in reduced memory usage and responding accurately to more diverse, noisy data. Additionally, bitwise operation-based error calculation enabled fewer operations during the runtime.

Although the proposed model had fast execution time and efficient memory and power usage, the model accuracy and other model evaluation metrics were slightly lower than conventional ML/DL approaches. Future works are needed to optimize the model for the real-world activity data to achieve better accuracy. However, it is important to preserve efficient resource consumption when improving the model accuracy since resource consumption efficiency is more important than a slight improvement in accuracy.

More research is needed to prove that MNIST data sets represent real-world activities. Compared to the MNIST data set, activity data in the real world may be more complex or challenging to distinguish between different labels. Moreover, the model's high accuracy obtained from using the MNIST dataset may not be obtained when using real-world activity data. Therefore, there is a slight chance that the model may not perform as well as expected on real-world data. However, due to the similarity between MNIST and activity data as an image and our model's ability with regard to image data classification, we firmly believe our model will perform well on real-world activity data classification. It would be best to use actual smart home activity data, but this might create complex ethical issues and would require legal consensus. We expect to obtain a large amount of real-world activity data for training, evaluating, and testing from smart homes in our future work. Based on

the real-world activity data, we expect to develop a preprocessing algorithm of raw activity data for the input data of our proposed model.

Additionally, our algorithm's training process is very efficient in terms of speed and memory usage. To clarify, the maximum memory usage without the training data set is 5.2 MB, including 6000 training images, and the training time is 584.5 ms on a normal PC(AMD Ryzen 5800X, 32 GB DDR4 Ram) for 6000 training data. We hope to optimize our training algorithm to be executed on edge in future work, thereby alleviating concerns about privacy in sending activity data to the server. We expect that it could be done by simply implementing the training algorithm suitable for the embedded device. Furthermore, due to the simple and scalable architecture of our training process, we anticipate our algorithm to be modified to perform on real-time machine learning problems. We believe it can be implemented immediately by developing an algorithm that determines the criterion for generating a new hyperplane by calculating the distance between the existing hyperplane and the input data. If the input data is relatively close to one of the existing hyperplanes, the input data will be classified to the corresponding hyperplane. On the other hand, if the new input data does not correspond to any of the hyperplanes, a new hyperplane will be created from the new input data. Therefore, we can accomplish real-time machine learning with little input data by slightly modifying the proposed algorithm.

7. Conclusions

In this study, we proposed an enhanced SVM algorithm for smart home activity data classification with improved performance and reduced power and memory usage. We demonstrated how smart home activity data corresponds to image data; therefore, we utilized the MNIST data set to verify our model's performance. In our proposed algorithm, training data for the same label are grouped into further detailed clusters, and then hyperplanes were generated by accumulating and thresholding each cluster based on a bitwise operation-based error function. We classified data at high speed and low power by the bitwise operation-based errors between input data and each hyperplane. We evaluated our method's performance on the 'Raspberry PI 3' and 'STM32 Discovery board' embedded systems. Compared to the LSVM that Tensorflow Lite implements, while the proposed algorithm had 82.2% overall accuracy, which is 3.6% lower than the LSVM, it improved performance by 46.3% and up to 50.4% depending on the system memory limitation, reduced peak memory usage to 15.41%, power consumption to 41.7%, and improved power per accuracy to 39.2%. Moreover, for the CNN model implemented by Tensorflow Lite, our model improved power consumption to 61.17%, memory usage to 15.41%, and performance up to 57.6%, while the accuracy was 12.4% lower. Therefore, we believe that our model will be suitable for real-world smart homes since the activity classification model is executed repeatedly in a resource-constrained device. Furthermore, because the training process was also high speed and memory efficient, it is anticipated that the proposed algorithm's training process could be executed on edge and may be extended to perform real-time machine learning due to its fast run time and simple, scalable architecture. Meanwhile, more research is needed to prove that the MNIST dataset can be used as a substitute for real-world activity data.

Author Contributions: J.C. designed, implemented and analyzed the entire model architecture; M.K. analyzed power consumption and performance; D.P. proposed the fundamental concept of the proposed methods and devoted his roles as the corresponding author. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the BK21 FOUR project funded by the Ministry of Education, Korea (4199990113966, 10%), and the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF2019R1A2C2005099, 10%), and Ministry of Education (NRF2018R1A6A1A03025109, 10%), and Institute of Information & communication Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (no. 2021-0-00944, Metamorphic approach of unstructured validation/verification for analyzing binary code, 70%), and the EDA tool was supported by the IC Design Education Center (IDEC), Korea.

Conflicts of Interest: The authors declare no conflicts of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

ML	Machine Learning
HAR	Human Activity Recognition
PR	Pattern Recognition
DNN	Deep Neural Network
CNN	Convolutional Neural Network
SAE	Stacked Autoencoder
RNN	Recurrent Neural Network
MNIST	Modified National Institute of Standards and Technology database
SVM	Support Vector Machine
LSVM	Linear Support Vector Machine
XOR	Exclusive Or
POPCOUNT	Population Count

References

1. Silverio-Fernández, M.; Renukappa, S.; Suresh, S. What is a smart device? A conceptualisation within the paradigm of the internet of things. *Vis. Eng.* **2018**, *6*, 3. [CrossRef]
2. Lee, S.; Park, D. Adaptive ECG Signal Compression Method Based on Look-ahead Linear Approximation for Ultra Long-Term Operating of Healthcare IoT Devices (SCI). *Hum. Centric Comput. Inf. Sci.* **2021**, *11*, 30. [CrossRef]
3. Lee, D.; Lee, S.; Oh, S.; Park, D. Energy-Efficient FPGA Accelerator with Fidelity-Controllable Sliding-Region Signal Processing Unit for Abnormal ECG Diagnosis on IoT Edge Devices (SCI). *IEEE Access* **2021**, *9*, 122789–122800. [CrossRef]
4. Casale, P.; Pujol, O.; Radeva, P. Human Activity Recognition from Accelerometer Data Using a Wearable Device. In *Pattern Recognition and Image Analysis*; Vitrià, J., Sanches, J.M., Hernández, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 289–296.
5. Elmenreich, W. *An Introduction to Sensor Fusion*; Vienna University of Technology: Vienna, Austria, 2002; Volume 502, pp. 1–28.
6. Kang, M.; Park, D. Remote Monitoring Systems of Unsafe Software Execution using QR Code-based Power Consumption Profile for IoT Edge Devices. In Proceedings of the 2021 International Conference on Electronics, Information, and Communication (ICEIC), Jeju, Korea, 31 January–3 February 2021; pp. 1–4. [CrossRef]
7. Seok, M.G.; Park, D. A Novel Multi-Level Evaluation Approach for Human-Coupled IoT Applications (SCI). *J. Ambient. Intell. Humaniz. Comput.* **2020**, 1–15. [CrossRef]
8. Risteska Stojkoska, B.L.; Trivodaliev, K.V. A review of Internet of Things for smart home: Challenges and solutions. *J. Clean. Prod.* **2017**, *140*, 1454–1464. [CrossRef]
9. Chan, M.; Estève, D.; Escriba, C.; Campo, E. A review of smart homes—Present state and future challenges. *Comput. Methods Programs Biomed.* **2008**, *91*, 55–81. [CrossRef]
10. Fortino, G.; Giordano, A.; Guerrieri, A.; Spezzano, G.; Vinci, A. A Data Analytics Schema for Activity Recognition in Smart Home Environments. In *Ubiquitous Computing and Ambient Intelligence. Sensing, Processing, and Using Environmental Information*; García-Chamizo, J.M., Fortino, G., Ochoa, S.F., Eds.; Springer: Cham, Swizerland, 2015; pp. 91–102.
11. Bing, K.; Fu, L.; Zhuo, Y.; Yanlei, L. Design of an Internet of Things-based smart home system. In Proceedings of the 2011 2nd International Conference on Intelligent Control and Information Processing, Harbin, China, 25–28 July 2011; Volume 2, pp. 921–924. [CrossRef]
12. Lee, D.; Moon, H.; Oh, S.; Park, D. mIoT: Metamorphic IoT Platform for On-Demand Hardware Replacement in Large-Scaled IoT Applications. *Sensors* **2020**, *20*, 3337. [CrossRef]
13. Park, D.; Youn, J.M.; Cho, J. A Low-Power Microcontroller with Accuracy-Controlled Event-Driven Signal Processing Unit for Rare-Event Activity-Sensing IoT Devices. *J. Sens.* **2015**, *2015*, 809201. [CrossRef]
14. Memos, V.A.; Psannis, K.E.; Ishibashi, Y.; Kim, B.G.; Gupta, B. An Efficient Algorithm for Media-based Surveillance System (EAMSuS) in IoT Smart City Framework. *Future Gener. Comput. Syst.* **2018**, *83*, 619–628. [CrossRef]
15. Sliwa, B.; Piatkowski, N.; Wietfeld, C. LIMITS: Lightweight Machine Learning for IoT Systems with Resource Limitations. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–7. [CrossRef]
16. Banbury, C.R.; Reddi, V.J.; Lam, M.; Fu, W.; Fazel, A.; Holleman, J.; Huang, X.; Hurtado, R.; Kanter, D.; Lokhmotov, A.; et al. Benchmarking TinyML systems: Challenges and direction. *arXiv* **2020**, arXiv:2003.04821.
17. Wang, J.; Chen, Y.; Hao, S.; Peng, X.; Hu, L. Deep Learning for Sensor-based Activity Recognition: A Survey. *arXiv* **2019**, arXiv:1707.03502.

18. Chavarriaga, R.; Sagha, H.; Calatroni, A.; Digumarti, S.T.; Tröster, G.; Millán, J.D.R.; Roggen, D. The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognit. Lett.* **2013**, *34*, 2033–2042. [CrossRef]
19. Barsocchi, P.; Cassará, P.; Giorgi, D.; Moroni, D.; Pascali, M.A. Computational Topology to Monitor Human Occupancy. *Proceedings* **2018**, *2*, 99. [CrossRef]
20. Hayashi, T.; Nishida, M.; Kitaoka, N.; Takeda, K. Daily activity recognition based on DNN using environmental sound and acceleration signals. In Proceedings of the 2015 23rd European Signal Processing Conference (EUSIPCO), Nice, France, 31 August–4 September 2015; pp. 2306–2310. [CrossRef]
21. Bulling, A.; Blanke, U.; Schiele, B. A Tutorial on Human Activity Recognition Using Body-Worn Inertial Sensors. *ACM Comput. Surv.* **2014**, *46*, 1–33. [CrossRef]
22. Walse, K.H.; Dharaskar, R.V.; Thakare, V.M. PCA Based Optimal ANN Classifiers for Human Activity Recognition Using Mobile Sensors Data. In *Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems*; Springer: Cham, Swizerland, 2016; Volume 1.
23. Yang, J.B.; Nguyen, M.N.; San, P.P.; Li, X.L.; Krishnaswamy, S. Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition. In Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, Buenos Aires, Argentina, 25–31 July 2015; pp. 3995–4001 .
24. Ha, S.; Yun, J.M.; Choi, S. Multi-modal Convolutional Neural Networks for Activity Recognition. In Proceedings of the 2015 IEEE International Conference on Systems, Man, and Cybernetics, Hong Kong, China, 9–12 October 2015; pp. 3017–3022. [CrossRef]
25. Bengio, Y. Deep Learning of Representations: Looking Forward. In *Statistical Language and Speech Processing*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 1–37. [CrossRef]
26. Sathyanarayana, A.; Joty, S.R.; Fernández-Luque, L.; Ofli, F.; Srivastava, J.; Elmagarmid, A.K.; Taheri, S.; Arora, T. Impact of Physical Activity on Sleep: A Deep Learning Based Exploration. *arXiv* **2016**, arXiv:1607.07034.
27. Ha, S.; Choi, S. Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors. In Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 24–29 July 2016; pp. 381–388. [CrossRef]
28. Mouhammed, B.S.; Artoli, A.M.; Al-Muhtadi, J. An effective deep autoencoder approach for online smartphone-based human activity recognition. *Int. J. Comput. Sci. Netw. Secur.* **2017**, *17*, 160–165.
29. Inoue, M.; Inoue, S.; Nishida, T. Deep recurrent neural network for mobile human activity recognition with high throughput. *Artif. Life Robot.* **2018**, *23*, 173–185. [CrossRef]
30. Yao, S.; Hu, S.; Zhao, Y.; Zhang, A.; Abdelzaher, T. DeepSense: A Unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing. In Proceedings of the 26th International Conference on World Wide Web; International World Wide Web Conferences Steering Committee: Republic and Canton of Geneva, WWW '17, Perth, Australia, 3–7 April 2017. [CrossRef]
31. Agarwal, P.; Alam, M. A Lightweight Deep Learning Model for Human Activity Recognition on Edge Devices. *Procedia Comput. Sci.* **2020**, *167*, 2364–2373. [CrossRef]
32. Moon, H.; Park, D. Efficient On-Demand Hardware Replacement Platform toward Metamorphic Functional Processing in Edge-Centric IoT Applications (SCI). *Electronics* **2021**, *10*, 2088. [CrossRef]
33. LeCun, Y.; Cortes, C. MNIST Handwritten Digit Database. 2010. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 1 September 2021).
34. Tabassum, M.; Kosinski, T.; Lipford, H.R. "I don't own the data": End User Perceptions of Smart Home Device Data Practices and Risks. In Proceedings of the Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019), Santa Clara, CA, USA, 11–13 August 2019; pp. 435–450.
35. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
36. Widasari, E.R.; Tanno, K.; Tamura, H. Automatic Sleep Disorders Classification Using Ensemble of Bagged Tree Based on Sleep Quality Features. *Electronics* **2020**, *9*, 512. [CrossRef]
37. MacQueen, J.B. Some Methods for Classification and Analysis of MultiVariate Observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*; Cam, L.M.L., Neyman, J., Eds.; University of California Press: Berkeley, CA, USA, 1967; Volume 1, pp. 281–297.
38. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
39. David, R.; Duke, J.; Jain, A.; Reddi, V.J.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Regev, S.; et al. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. *arXiv* **2020**, arXiv:2010.08678.
40. Nethercote, N.; Seward, J. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. *SIGPLAN Not.* **2007**, *42*, 89–100. [CrossRef]
41. Kang, M.; Park, D. Lightweight Microcontroller with Parallelized ECC-based Code Memory Protection Unit for Robust Instruction Execution in Smart Sensors (SCI). *Sensors* **2021**, *21*, 5508. [CrossRef] [PubMed]
42. Chang, J.; Kim, B.; Mun, C.; Lee, D.; Kwak, J.; Park, D.; Jeong, Y. Efficient Hyperplane Generation Techniques for Human Activity Classification in Multiple-Event Sensors Based Smart Home. *IEMEK J. Embed. Syst. Appl.* **2019**, *14*, 277–286.