Enhancing Efficiency in HAR Models: NAS Meets Pruning

Yexu Zhou, Tobias King, Yiran Huang, Haibin Zhao, Till Riedel, Tobias Röddiger, Michael Beigl
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{yexu.zhou, tobias.king, yiran.huang, haibin.zhao, till.riedel, tobias.roeddiger, michael.beigl}@kit.edu

***Abstract*—Real-time monitoring of human activities using wearable devices often requires the deployment of machine learning models on resource-constrained edge devices. State-of-the-art Human Activity Recognition models suffer from excessive size and complexity. Furthermore, our systematic analysis reveals that even worse, the computational cost and model size of most SOTA HAR models escalate significantly with increasing sensor channels. With advances in sensor technology that make it easier to scale sensor deployments that capture human activities, addressing this challenge becomes critical for practical applicability. In this work, we propose an integrated neural architecture search framework to further lighten HAR models. The proposed framework simultaneously selects and reduces the number of sensor channels, prunes filters, and decreases the temporal dimensions while training the model on optimized hardware. This results in smaller and less complex models. Experiments on three HAR datasets demonstrate that our framework outperforms two state-of-the-art pruning methods in reducing model size and complexity, while achieving superior performance. Furthermore, we successfully applied our proposed framework to the deployment of a HAR model on a microcontroller, highlighting its feasibility for real-world implementation.**

***Index Terms*—human activity recognition, automated machine learning, hardware-aware neural architecture search**

I. Introduction

Pervasive sensing systems are designed to sense, collect and analyze environmental data with minimal human intervention. Within this expansive domain, Human Activity Recognition (HAR) emerges as a prominent task [1]. HAR primarily utilizes data obtained from (multiple) Inertial Measurement Units (IMUs) to identify and categorize human activities.

Recently, neural network models have outperformed traditional machine learning methods in the field of HAR [2]–[4]. This success can be attributed to the ability of neural networks to automate feature learning and extraction. To enhance the efficiency of these models, various deep learning architectures have been proposed. However, a significant challenge arises from the excessive size and complexity of these models, often surpassing the hardware limitations of edge devices, thereby hindering their deployment in real-world applications.

Furthermore, our investigations have revealed a direct correlation: as the number of sensor channels increases, there is a corresponding increase in both model size and complexity. For example, as detailed in Figure 2, when the number of sensor channels doubles, the necessary memory and computational demands also almost double. This scenario poses a significant challenge when dealing with data collected from multiple IMU sources, resulting in oversized models.

To address this issue, we conducted a comprehensive analysis of the structure, design, and computational complexity of the model. Most HAR models primarily employ individual convolutional layers at the initial stage, consuming a significant portion of computational resources. Based on these insights, we have developed a NAS framework tailored for the lightweight optimization of HAR models. The framework filters sensor channels, prunes filters per layer, reduces the temporal dimension, and optimizes the model structure in an end-to-end manner.

- We systematically analyze the design of state-of-the-art HAR models, illustrating the causes of oversized models and excessive model complexity.
- Based on these findings, we propose a NAS framework for the lightweight optimization of HAR models.
- We demonstrate that our proposed NAS framework significantly reduces the size and computational requirements while achieving better performance than SOTA pruning algorithms on the three benchmark datasets.
- Finally, we evaluate the framework on an embedded system we use for wearable prototyping, the STMicroelectronics STM32 Nucleo-144 Development Board (NUCLEO-L552ZE-Q) equipped with an STM32L552 microcontroller unit (MCU), to demonstrate its real-world applicability and efficiency.

II. Related Work

A wide variety of complex neural network architectures have been proposed to enhance feature extraction capabilities and achieve SOTA performance. These architectures include hybrid models that combine Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs)[5]–[7], as well as attention-based models[8]. The work of Ismail et al. [9] has employed NAS approaches to discover effective neural network models. Although these models have achieved exceptional performance, they often prove impractical for deployment on edge devices due to their size.

Subsequent efforts, such as [2], have focused on designing lightweight models through expert knowledge. However,

This work was partially funded by the German Ministry for Research and Education as part of SDI-S (Grant 01IS22095A) as well as by the Ministry of Science, Research and the Arts Baden-Wuerttemberg as part of the SDSC-BW project and by the Carl-Zeiss-Foundation as part of the JuBot project.

adapting the deployment for different application scenarios or hardware constraints still requires expert knowledge. Other works [10], [11], have utilized hardware-aware NAS techniques to obtain lightweight models, considering hardware limitations during optimization. Although these optimized models have yielded more favorable results, they are still too large for deployment on devices with extremely limited resources, such as MCUs.

A. Overview of HAR Model Architecture

DeepConvLSTM	**1D Individual Convolutional Block**	x	**Flatten**	LSTM
Attend and Discriminate	**1D Individual Convolutional Block**	Cross-Channel Interaction Encoder	**Flatten**	Attentional GRU Layer
IF-Conv Transformer	Complementary Filter + **1D Individual Convolutional Block**	Cross-Channel Interaction Encoder	**Flatten** + 1D Convolution Layer	Transformer Block
DeepSense	**1D Individual Convolutional Block**	x	**Flatten** + Merge Convolutional block	GRU
TinyHAR	**1D Individual Convolutional Block**	Cross-Channel Interaction Encoder	**Flatten** + Fully Connected Layer	Attentional LSTM Layer
AttnSense	**1D Individual Convolutional Block**	x	Attention-fusion Subnet	Attentional GRU Layer
GlobalFusion	**1D Individual Convolutional Block**	Spatial Convolutional Block	Modality Convolutional Block	GRU
ALAE-TAE-CutMix	**1D Individual Convolutional Block**	Adaptive Latent Attention Encoder	**Flatten**	Attentional LSTM Layer

Fig. 1. Overview of the architecture design of HAR models.

The challenge in predicting activities lies in precisely capturing local context, multi-modal interactions, and global temporal information from multidimensional sensor data [2], [12]. As illustrated in Figure 1, the design of feature extraction can be segmented into four stages [2]: extraction of local context, sensor information interaction, sensor information fusion, and temporal information extraction. This architecture design accounts for the characteristics of the data. In line with these design principles, as illustrated in the table in Figure 1, the architecture of most SOTA HAR models, such as Attend-and-Discriminate [4], If-ConvTransformer [5], DeepSense [3], AttenSense [12], TinyHar [2], AttenSense [12], GlobalFusion [8] and ALAE-TAE [1] models, can be categorized into these stages. The distinction between these models lies in their approach to sensor fusion and temporal information extraction. However, nearly all these networks initially utilize identical individual convolution blocks in the initial stage for local context extraction, acknowledging that each sensor channel uniquely captures information for specific activities.

III. Preliminaries

A. Problem Definition

The aim of HAR is to use a classification model to map segments of sensor readings to specific activity classes. Consider $\boldsymbol{X}_0 \in \mathbb{R}^{f_0 \times c_0 \times s_0}$ as the input segment, where s_0 indicates the length of the input segment, and c_0 represents the count of sensor channels. Given the unprocessed nature of the raw data, the feature dimension is initially set at $f_0 = 1$. In this study, the subscript notation $i = 0, 1, 2, ..., L$ represents the specific model layer associated with a variable, with $i = 0$ explicitly indicating the input layer of the model.

B. HAR Model Complexity and Size

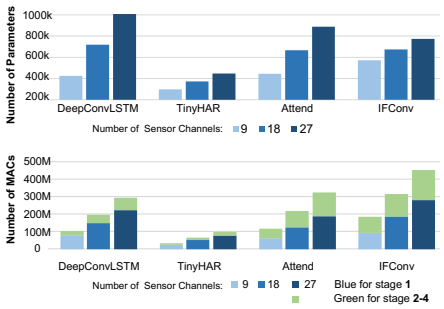

Fig. 2. Impact of the number of sensor channels.

In this research, we focused on four distinct models and evaluated their model complexity at each stage. For measuring model complexity, we used Multiply-Accumulate Operations (MACs) as the metric. To highlight the complexity of the initial stage, we divided the four stages into two parts: the first part solely includes the first stage, while the second part comprises the subsequent stages. This evaluation was carried out in the context of the UCI-HAR data scenario [13], which uses a single IMU, equal to nine sensor channels, for data acquisition. Furthermore, we modified the count of sensor channels to mimic scenarios that involve the use of additional IMUs. This adjustment facilitates the check of how the number of sensor channels affects both the size and complexity.

As shown in Figure 2, the first stage accounts for the majority of MACs in all models analyzed. This is primarily due to the fact that the temporal dimension of the HAR data is significantly larger than the other dimensions. The first stage operates directly on these original dimensions, leading to a high computational intensity. This observation suggests that an effective way to reduce the complexity of the model is to shorten the length of the temporal dimension in this convolutional stage. A significant reduction in the temporal dimension in the first stage can also benefit subsequent stages. For example, the cross-channel interaction encoder [4] and various temporal information extraction blocks, which operate along the temporal dimension, would require less computation following this reduction. Furthermore, we found that the number of convolutional filters in the first stage of these HAR models is generally uniformly predefined [4], [7]. Thus, reducing the number of filters in this stage could further decrease the complexity of the model.

Surprisingly, we discovered that increasing the number of IMUs leads to a dramatic increase in both the model's parameters and complexity. Specifically, when the number of sensor channels increases from 9 to 18, the size of the model and the computational cost nearly double. This increase is attributed to the individual convolutions performed on the expanded sensor channels. Another factor contributing to this increase is the model's use of a flatten operation followed by a Fully Connected (FC) layer for feature fusion. Alternatively,

the model directly feeds the vectorized features into the subsequent temporal extraction stage, which also includes many FC layers. It is well known that the number of parameters and model complexity of an FC layer is proportional to its input and output dimensions. Therefore, an increase in sensor channels enlarges the input dimension for these FC layers, significantly enlarging its computational demands.

In summary, to make the model more lightweight, the most effective strategies include optimizing the first stage by aiming to reduce the temporal dimension size as much as possible, selecting efficient sensor channels, and decreasing the number of filters in each convolutional layer.

IV. Methodology

Based on the previous findings, we have designed an NAS framework that can be applied to any HAR model featuring an individual convolution as the first stage.

A. Sensor Channel and Convolutional Filter Pruning

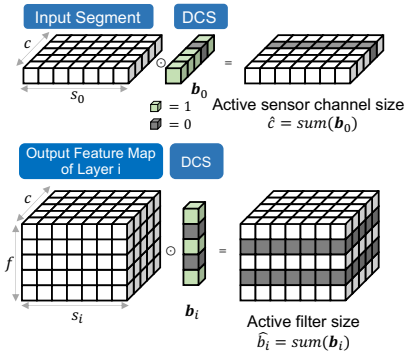

Fig. 3. DSC layer for the input layer and convolutional layer.

To facilitate the reduction of both the sensor channels and the convolutional filters, a pruning strategy is integrated into the framework. Specifically, a Differentiable Channel-wise Scaling layer (DCS)[14] (We've changed its name in the original paper for ease of understanding) is inserted at the outset of the model and subsequently at the end of each convolution, as depicted in Figure3. The mathematical formulation of this process is represented as follows,

$$\mathbf{X}_i = \mathbf{b}_i \odot \mathbf{X}_i \quad i = 0 \tag{1}$$

$$\mathbf{X}_{i+1} = \mathbf{b}_i \odot (\mathbf{X}_{i-1} * \mathbf{w}_i) \quad i = 1, 2, \cdots, L \tag{2}$$

In this equation, the symbol $*$ denotes the convolution operation, while $\odot$ represents element-wise multiplication. When $i = 0$, which indicates the input layer, there is no convolutional layer. The initial DCS layer is crucial to prune the sensor channels within the original data, with $\mathbf{b}_0 \in \mathbb{R}^{1\times c_0\times 1}$. In contrast, subsequent DCS layers focus on pruning filters in the convolutions preceding them, with $\mathbf{b}_i \in \mathbb{R}^{f\times 1\times 1}, i = 1, 2, \cdots, L$. Each value of the parameter $\mathbf{b}_i$ acts as an indicator of the importance of the corresponding sensor or filter channel. A value of 1 signifies importance, ensuring the retention of the corresponding feature map portion for subsequent layers. On the contrary, a value of 0 indicates irrelevance, leading to the nullification of the corresponding feature map section. Thus, the tasks of filtering the sensor channel and pruning the filter become a binary optimization problem dependent on the parameter $\mathbf{b}_i$.

To address the issue of gradient back-propagation related to this parameter, the Straight-Through Estimator (STE) [15] technique was employed. The forward and backward propagation processes are defined as follows,

$$Forward: \mathbf{b}_i = \begin{cases} 0 & \mathbf{v}_i \le thres \\ 1 & \mathbf{v}_i > thres \end{cases} \tag{3}$$

$$Backward: \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{b}_i} \tag{4}$$

In this context, $\mathbf{v}_i$ is a continuous parameter, while $\mathbf{b}_i$ is binarized, derived from $\mathbf{v}_i$. The threshold value $thres$ is a hyper-parameter, set at 0.5 in this study. Through the integration of DCS layers and the STE method, both the model parameters $\mathbf{w}_i$ and the policy pruning parameters $\mathbf{v}_i$ can be trained simultaneously.

B. Search Space

As mentioned in the previous section, in addition to pruning sensor and filter channels, it is imperative to condense the temporal dimension as effectively as possible. To this end, we have devised a search space, as shown in Figure 4. This space is structured through the intersection of convolutional layers with strides of 2 and those with strides of 1. The number of convolutional layers with a stride of 2 is determined by the length of the inputs. To avoid excessive reduction in the temporal dimension, a specific strategy is employed. We define the number of layers with a convolution stride of 2 as follows: $N = \lfloor \log_2 \left(\frac{s_0}{4}\right) \rfloor$. The denominator 4 ensures that, when all convolutional layers with a stride of 2 are utilized, the resulting size in the temporal dimension will be at least 4.

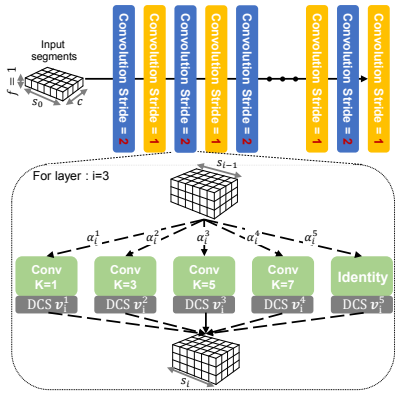

Fig. 4. Search space.

Regarding each convolution layer, the available operations, as presented in Figure 4, include convolutions with kernel sizes of 1, 3, 5, and 7, along with an identity operator. Each candidate operator is associated with a architecture parameter $\alpha_i^j, j = 1, 2, 3, 4, 5$. When a layer is designated as an identity layer, it contributes to reducing the total number of layers in the model. Additionally, the use of layers with a stride of 2 is crucial in facilitating the compression of the temporal

dimension. For selecting operators for each layer, we adopt the DARTS methodology [16] combined with the Gumbel-Max trick approach [17]. This approach involves sampling an operator during forward inference as follows.

$$\boldsymbol{p}_i = \text{Onehot\_Encoding}\left(\underset{j}{\text{argmax}}\left(g^j + \log\left(\alpha_i^j\right)\right)\right) \quad (5)$$

where g^i are independent samples drawn from a standard Gumbel distribution ($g^i \sim \text{Gumbel}\,(0,1)$). $\boldsymbol{p}_i$ is a one-hot encoded vector representation, with a probability proportional to the value of α_i^j. The forward process is as follows:

$$\boldsymbol{X}_{i+1} = \sum_{j=1}^{5} \boldsymbol{p}_i^j op_i^j\left(\boldsymbol{X}_{i-1}\right) \quad (6)$$

At any given instance, only one operator is applied. During gradient back-propagation, we compute the gradient of the argmax in equation 5 through a differentiable approximation as follows:

$$\text{G\_Softmax}\left(\alpha_i^j;\boldsymbol{\alpha}\right) = \frac{\exp\left(\left(\log\left(\alpha_i^j\right) + g_i^j\right)/\tau\right)}{\sum_{k=1}^{5}\exp\left(\left(\log\left(\alpha_i^k\right) + g_i^k\right)/\tau\right)} \quad (7)$$

Here, τ is the temperature parameter controlling the approximation's fidelity to discrete one-hot vectors. Consequently, this allows the model to be trained with discrete operations, using equations 5 and 6 for the forward pass and the differentiable equation 7 for gradient backpropagation.

C. Optimization

following the work [16], the entire optimization process can be mathematically defined as follows,

$$\min_{\alpha}\ \mathcal{L}_{val}\left(\boldsymbol{w}^*\left(\boldsymbol{\alpha}\right),\boldsymbol{v}^*\left(\boldsymbol{\alpha}\right),\boldsymbol{\alpha}\right) \quad (8)$$

$$s.t.\ \ \boldsymbol{w}^*\left(\boldsymbol{\alpha}\right),\boldsymbol{v}^*\left(\boldsymbol{\alpha}\right) = \underset{\boldsymbol{w},\boldsymbol{v}}{argmin}\ \ \mathcal{L}_{train}\left(\boldsymbol{w},\boldsymbol{v},\boldsymbol{\alpha}\right) \quad (9)$$

where $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ denote the training and validation loss, respectively. This is a typical bi-level optimization problem [16], with $\boldsymbol{\alpha}$ as the upper-level variable and the model weight $\boldsymbol{w}$ and learnable scaling factors $\boldsymbol{v}$ as the lower-level variable. The optimization of these two parameters will follow the iterative optimization procedure used in the DARTS.

where $\mathcal{L}train$ and $\mathcal{L}_{val}$ denote training and validation losses, respectively. This formulation represents a typical bi-level optimization problem [16], with $\boldsymbol{\alpha}$ as the upper-level variable, and the model weights $\boldsymbol{w}$ and the learnable scaling factors $\boldsymbol{v}$ as the lower-level variables. The optimization of these parameters will follow the iterative optimization procedure used in DARTS. In equation 8, the loss function comprises the cross-entropy loss $\mathcal{L}crossentropy$ and hardware-aware constraints as follows:

$$\mathcal{L} = \mathcal{L}_{crossentropy} + \beta_1\mathcal{L}_{complexity} + \beta_2\mathcal{L}_{memory} \quad (10)$$

The hardware-aware constraints include the model complexity constraint and peak memory usage constraints. β_1 and β_2 are trade-off parameters that adjust the impact of efficiency and memory usage on overall loss. During the total loss minimization process, a policy is learned that decides whether to remove or keep a layer, as well as alter its number of filters and sensor channels. The model complexity loss $\mathcal{L}complexity$ is analytically estimated as follows:

$$\mathcal{L}_{complexity} = \sum_{i=1}^{L}\sum_{j}^{5}\mathbf{p}_i^j\left|\hat{b}_{i-1}\cdot\hat{b}_i^j\cdot\hat{c}_0\frac{\cdot s_{i-1}}{t_i}\cdot k_i^j\cdot 1\right|^2 \quad (11)$$

For each layer i, the computational complexity of the selected op_i^j is included in the L2 norm regularization. $\hat{\mathbf{b}}_i^j = sum\left(\mathbf{b}_i^j\right)$ denotes the active filter number of the selected operator, representing the output dimension. $\hat{\mathbf{b}}_{i-1}$ represents the input dimension. t_i denotes the stride of the current layer. k_i^j denotes the current kernel size. The size of the active sensor channel $\hat{c}_0 = sum\left(\hat{\mathbf{b}}_0\right)$ is taken from the first DCS layer for the input segment.

When an edge device has a very limited amount of SRAM, it is crucial to factor in peak memory usage into the training loss. To calculate the maximum memory consumption of an architecture, we use the analytical estimation method proposed in [18]. The memory required to perform an operation is estimated as follows:

$$mem\,(op) = mem\,(input) + mem\,(output) + extra\,(op) \quad (12)$$

To execute an operation in a neural network, both the input and output tensors of the operation need to be present in memory. Additionally, some operations, such as those of the CMSIS-NN kernel library, require extra memory for computation. The maximum memory consumption $PMem(\alpha,\mathbf{v})$ can be computed as the maximum memory requirement in all operations. The loss for peak memory consumption is defined as follows:

$$\mathcal{L}_{memory} = \begin{cases} log\left(\frac{PMem(\alpha,\mathbf{v})}{L\_Mem}\right), & if PMem\,(\alpha,\mathbf{v}) > L\_Mem \\ 0, & otherwise \end{cases} \quad (13)$$

A penalty is applied when peak memory usage exceeds the defined maximum memory limit L_Mem.

V. Experiments and Discussions

A. Experiment Setup

1) Datasets: We use three HAR benchmark datasets to validate the proposed framework and provide empirical evidence of its generalizability. These three datasets are chosen because of their great diversity in terms of the sensing modalities used and the activities to be recognized. The **Skoda** [19] dataset contains 10 manipulative gestures performed in a car maintenance scenario. The 30 signals from the 10 sensors attached to the right-hand body are used. All sensor signals are segmented by a sliding window of 196 readings (2 seconds) with 50% overlap. Following the work [1], the first 80% of the data is used for training, the next 10% for validation, and the remaining for testing. The UCI-HAR dataset [13] are collected from a group of 30 volunteers, who perform 6 activities while wearing a smartphone on the waist. The data are already segmented in sliding windows with fixed width of 2.56 s and

50% overlap. Following the default setup in [13], the data from user id 2,4,9,10,12,13,18,20,24 are used as a test dataset. The Motion Sense dataset [20] was collected from 24 volunteers who kept a phone in their front pocket and performed 6 daily activities. Following the suggestion [5] , a sliding window of 128 readings (2.56 s) with 10 reading overlap is adopted to segment the sensor signals, and trial id greater than 10 are used as test sets. The signals of each sensor are z-normalized.

2) HAR Models: To demonstrate the applicability of the proposed framework to various models, we consider three HAR models in our experiments: DeepConvLSTM (DCL), TinyHAR, and Attend-and-Discriminate (Attend). All three models utilize an individual convolution block at the initial stage. Therefore, we replace only the initial stage with our proposed searchable framework, while the structure of the subsequent stages remains consistent with the original works.

3) Compared Pruning Approaches: To make the model lightweight, pruning is a commonly used method. In addition to our proposed method, we also consider two SOTA pruning methods: AutoSlim [21] and PaS [14] as a baseline. We did not modify these two algorithms, so they can only prune the number of filters in each layer. However, our algorithm can prune not only the **S**ensor channels and the **F**ilters of each layer but also reduce the length of the **T**emporal dimension. Therefore, we have named our model SFTNAS.

4) Training&Evaluation Protocol: To optimize the model weights $\boldsymbol{w}$ and the scaling parameters $\mathbf{v}$, we use the Adam optimizer with default parameters and an initial learning rate of $\xi_w = 10^{-4}$. The learning rate decays by 0.9 after reaching a patience threshold of 10 epochs. The maximum number of training epochs is set to $epoch_{maximum} = 200$, and the batch size is set to 256. For the optimization of the architecture parameters $\boldsymbol{\alpha}$, we employ an additional Adam optimizer with a learning rate of $\xi_\alpha = 5\times10^{-3}$, momentum $\beta = (0.5, 0.999)$, and weight decay of 10^{-3}. The parameters $\boldsymbol{\alpha}$ are initialized to 10^{-3}. The temperature τ is initially set to $\tau_0 = 5.0$, with its decay equation being $\tau = \tau_0 \times \exp(-0.05 \times epoch)$. Given the imbalance in the HAR data, we use the macro F1 score ($F1_M$) as the evaluation metric.

B. Optimization Results

The three tables,I,II, and III, provide performance results for the algorithms applied to different HAR models across three datasets: Motion Sense, UCI-HAR, and SKODA. The algorithms with the best performance are highlighted in bold. From a general point of view, SFTNAS consistently stands out as the most effective algorithm in reducing model size across all datasets and models. It achieves significant reductions in the number of parameters compared to the original models. In addition, SFTNAS also proves to be highly efficient in terms of model complexity reduction, achieving the lowest MMACs across all combinations of datasets and models. This indicates that it can significantly improve the inference time. AutoSlim and PaS also contribute to reducing model size and complexity, but not as effectively as SFTNAS. Regarding the classification performance of the algorithms, SFTNAS helps all three HAR models to have the smallest model size and the least model complexity on the Motion Sense and UCI-HAR datasets, and achieves optimal results. Only on the SKODA dataset does SFTNAS achieve slightly lower results than the other two pruning algorithms. However, it is important to emphasize that the decrease in both model size and model complexity on the SKODA dataset is significant compared to the other two pruning algorithms. This is due to the fact that the proposed SFTNAS not only reduces the number of filters, but also reduces the size of the temporal dimension as well as performs the pruning of the sensor channel. One shortcoming is that the proposed SFTNAS can only optimize the initial stage, and the subsequent model structures are Fixed.

TABLE I
PERFORMANCE ON MOTION SENSE DATA SET

Model + Methods	Parameters	MMACs	$F1_m$
DCL	522.57 k	138.73	92.23
DCL + SFTNAS	**81.92 k**	**2.32**	**89.88**
DCL + AutoSlim	131.24 k	35.14	88.67
DCL + PaS	95.33 k	25.33	88.21
Attend	519.11 k	155.62	94.42
Attend + SFTNAS	**56.45 k**	**3.74**	**91.37**
Attend + AutoSlim	81.76 k	24.92	90.98
Attend + PaS	73.15 k	19.02	90.46
TinyHAR	322.76 k	47.17	92.12
TinyHAR + SFTNAS	**41.28 k**	**1.79**	**90.17**
TinyHAR + AutoSlim	50.03 k	7.69	89.21
TinyHAR + PaS	54.27 k	6.15	**90.20**

TABLE II
PERFORMANCE ON UCI-HAR DATA SET

Model + Methods	Parameters	MMACs	$F1_m$
DCL	424.26 k	105.95	92.51
DCL + SFTNAS	**33.74 k**	**1.86**	**92.49**
DCL + AutoSlim	46.21 k	11.72	91.83
DCL + PaS	37.89 k	9.64	92.04
Attend	445.38 k	120.55	94.07
Attend + SFTNAS	**41.11 k**	**1.37**	**93.29**
Attend + AutoSlim	48.62 k	13.59	92.88
Attend + PaS	39.89 k	11.2	92.71
TinyHAR	298.18 k	36.27	94.58
TinyHAR + SFTNAS	**13.96 k**	**0.767**	**94.36**
TinyHAR + AutoSlim	39.99 k	6.21	93.58
TinyHAR + PaS	21.34 k	3.56	94.19

TABLE III
PERFORMANCE ON SKODA DATA SET

Model + Methods	Parameters	MMACs	$F1_m$
DCL	1.11 M	534.44	91.12
DCL + SFTNAS	**61.66 k**	**3.54**	**90.01**
DCL + AutoSlim	173.93 k	83.72	88.45
DCL + PaS	187.93 k	70.42	89.33
Attend	962.0 k	592.41	92.39
Attend + SFTNAS	**63.87 k**	**5.03**	91.04
Attend + AutoSlim	139.17 k	90.44	91.23
Attend + PaS	150.82 k	87.65	**91.59**
TinyHAR	470.73 k	183.2	90.48
TinyHAR + SFTNAS	**38.01 k**	**4.27**	87.48
TinyHAR + AutoSlim	141.73 k	57.53	87.01
TinyHAR + PaS	104.38 k	43.01	**88.31**

C. Experimental deployment on Micro-controller

We also performed an evaluation on the NUCLEO-L552ZE-Q, equipped with a STM32L552 (Single core, ARM Cortex-M33, 80 MHz CPU clock, 512 kB flash, 256 kB SRAM), using the UCI-HAR dataset. Given the extreme limitations of this hardware, the three models, as well as the variants pruned by AutoSlim and PaS, could not be deployed on it.

To deploy the model on this hardware, we applied the proposed SFTNAS to TinyHAR. In addition to considering the model complexity during training, we also included peak memory in the training loss, with L_Mem set to 96 kB. After the model was trained, this optimized architecture was further fine-tuned using quantization-aware training, and later the resulting model was deployed to the MCU using int_8 quantization. Adjusting the size of β_1, we could obtain models with different computational complexities. We experimented with various values of β_1 and plotted the trade-off between the inference time and the performance of the resulting models in Figure 5 (a). This experiment demonstrates SFTNAS's ability to find architectures under different model complexity constraints. When higher model complexities are allowed, classification performance increases.

We further analyze the distribution of the number of filters per layer in the optimize models and plot the distribution in Figure 5 (b). As can be observed from the figure, there is an increasing trend in the number of filters as the number of layers increases. This trend can serve as a heuristic for the manual design of models.

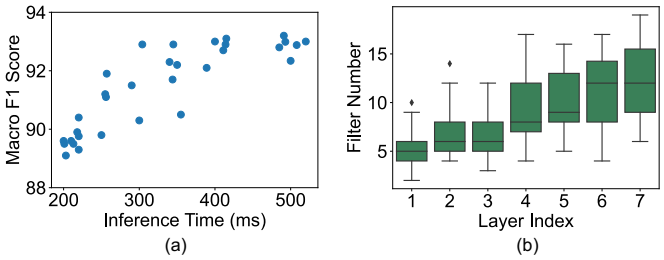

Fig. 5. (a) Trade-off between inference time and performance. (b) Distribution of optimized filter numbers.

VI. Conclusion

In this work, we have optimized the design of individual blocks in the HAR model by using NAS in combination with a pruning technique. This approach is designed to minimize the size of various dimensions, namely the sensor channel dimension, time dimension, and filter dimension, without significantly sacrificing the model's performance. It has been experimentally demonstrated that the proposed method can significantly reduce the size and complexity of the model while maintaining its performance. In many cases, the complexity is only about 2% of the original. This observation indirectly shows that reducing the complexity of the model in the first stage can also benefit subsequent computations. Finally, we demonstrate the usability of the proposed method by applying it on a microcontroller.

References

[1] N. Ahmad and H.-f. Leung, "Alae-tae-cutmix+: Beyond the state-of-the-art for human activity recognition using wearable sensors," in *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, IEEE, 2023, pp. 222–231.

[2] Y. Zhou, H. Zhao, Y. Huang, T. Riedel, M. Hefenbrock, and M. Beigl, "Tinyhar: A lightweight deep learning model designed for human activity recognition," in *Proceedings of the 2022 ACM International Symposium on Wearable Computers*, 2022, pp. 89–93.

[3] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 351–360.

[4] A. Abedin, M. Ehsanpour, Q. Shi, H. Rezatofighi, and D. C. Ranasinghe, "Attend and discriminate: Beyond the state-of-the-art for human activity recognition using wearable sensors," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 1, pp. 1–22, 2021.

[5] Y. Zhang, L. Wang, H. Chen, A. Tian, S. Zhou, and Y. Guo, "If-convtransformer: A framework for human activity recognition using imu fusion and convtransformer," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 2, pp. 1–26, 2022.

[6] Y. Zhou, M. Hefenbrock, Y. Huang, T. Riedel, and M. Beigl, "Automatic remaining useful life estimation framework with embedded convolutional lstm as the backbone," in *ECML PKDD 2020*, Springer, 2021.

[7] F. J. Ordóñez and D. Roggen, "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[8] S. Liu, S. Yao, J. Li, *et al.*, "Giobalfusion: A global attentional deep learning framework for multisensor information fusion," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*,

[9] W. N. Ismail, H. A. Alsalamah, M. M. Hassan, and E. Mohamed, "Auto-har: An adaptive human activity recognition framework using an automated cnn architecture design," *Heliyon*, 2023.

[10] X. Wang, X. Wang, T. Lv, L. Jin, and M. He, "Harnas: Human activity recognition based on automatic neural architecture search using evolutionary algorithms," *Sensors*,

[11] X. Wang, M. He, L. Yang, H. Wang, and Y. Zhong, "Human activity recognition based on an efficient neural architecture search framework using evolutionary multi-objective surrogate-assisted algorithms," *Electronics*, vol. 12, p. 50, 2022.

[12] H. Ma, W. Li, X. Zhang, S. Gao, and S. Lu, "Attnsense: Multi-level attention mechanism for multimodal human activity recognition.," in *IJCAI*, 2019, pp. 3109–3115.

[13] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. L. Reyes-Ortiz, *et al.*, "A public domain dataset for human activity recognition using smartphones.," in *Esann*, vol. 3, 2013, p. 3.

[14] Y. Li, P. Zhao, G. Yuan, X. Lin, Y. Wang, and X. Chen, "Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization," *arXiv preprint arXiv:2206.01198*, 2022.

[15] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[16] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[17] E. J. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*. 1948, vol. 33.

[18] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "Hardware-aware neural architecture search: Survey and taxonomy.," in *IJCAI*, 2021, pp. 4322–4329.

[19] P. Zappi, C. Lombriser, T. Stiefmeier, *et al.*, "Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection," in *EWSN 2008*.

[20] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi, "Mobile sensor data anonymization," in *Proceedings of the international conference on internet of things design and implementation*, 2019, pp. 49–58.

[21] J. Yu and T. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," *arXiv preprint arXiv:1903.11728*, 2019.