RESEARCH ARTICLE

AutoGCN-Toward Generic Human Activity Recognition With Neural Architecture Search

FELIX TEMPEL [1], **ESPEN ALEXANDER F. IHLEN** [2], **AND INGA STRÜMKE** [1]

[1]Faculty of Informatics, Norwegian University of Science and Technology, 7491 Trondheim, Norway
[2]Faculty of Medicine and Health Sciences, Norwegian University of Science and Technology, 7491 Trondheim, Norway

Corresponding author: Felix Tempel (felix.e.f.tempel@ntnu.no)

ABSTRACT This paper introduces AutoGCN, a generic Neural Architecture Search (NAS) algorithm for Human Activity Recognition (HAR) using Graph Convolution Networks (GCNs). HAR has enjoyed increased attention due to advances in deep learning, increased data availability, and enhanced computational capabilities. Concurrently, GCNs have shown promising abilities in modeling relationships between body key points in a skeletal graph. Typically, domain experts develop dataset-specific GCN-based methods, which limits their applicability beyond the specific context. AutoGCN seeks to address this limitation by simultaneously searching for the ideal hyperparameters and architecture combination within a versatile search space using a reinforcement controller while balancing optimal exploration and exploitation behavior with a knowledge reservoir during the search process. We conduct extensive experiments on two large datasets focused on skeleton-based action recognition to assess the proposed algorithm's performance. Our experimental results demonstrate the effectiveness of AutoGCN in constructing optimal GCN architectures for HAR, outperforming conventional NAS and GCN methods, as well as random search. These findings highlight the significance of a diverse search space and an expressive input representation to achieve good model performance and generalizability.

INDEX TERMS Neural architecture search, human activity recognition, graph convolution networks, AutoML.

I. INTRODUCTION

Human Activity Recognition (HAR), the process of identifying and categorizing human activities, has witnessed a surge in interest in recent years. This momentum is attributed to advancements in deep learning techniques, the increased availability of data, and enhanced computational capabilities [1]. HAR has applications in a wide range of domains, including video surveillance, human-computer interaction, and healthcare, highlighting its significance and broad impact [1], [2], [3], [4]. Within the realm of HAR, the models aim to capture representations of humans using a

The associate editor coordinating the review of this manuscript and approving it for publication was Ganesh Naik [ID].

skeleton graph built of body key points [1]. In the context of HAR, the traditional feature extraction method relies on manually crafted descriptors to capture fundamental attributes [5], [6]. Another approach involves formulating the problem as a deep learning task in Euclidean space. This approach uses the capabilities of Convolutional Neural Networks and Recurrent Neural Networks associated with recognition tasks. However, approaches involving these architectures have a significant shortcoming in modeling the inherent structures in skeletons, as the input data is serialized [1]. Recently, Graph Neural Networks (GNNs), specifically Graph Convolution Networks (GCNs), have emerged as a cutting-edge solution in HAR [7]. GCNs have achieved remarkable performance by leveraging the relationships between different segments and joints, treating

these as nodes and edges in an adjacency matrix, making convolutional operations possible [1], [7].

While GCN-based methods have shown promising results, they are primarily developed by domain experts and often tailored to specific datasets through a trial-and-error process. This limits their potential impact, as applying them to a broader range of problems can be challenging or when domain knowledge is unavailable. One approach to overcoming those challenges is automatically building task-specific deep learning models using Neural Architecture Search (NAS) [8]. By employing NAS, researchers can discover optimal architectures from a defined search space through a search procedure, enhancing the overall performance and generalizability of the resulting models.

In this work, we propose a generic NAS algorithm, Auto-GCN, that dynamically learns both hyper- and architecture parameters to construct a framework that maximally fits the HAR task. We present the following contributions:

- A generic approach to construct a GCN architecture for HAR with NAS, which can concurrently optimize hyperparameters and the model architecture. Additionally, the approach can be applied to different datasets within HAR and performs better than random search and other NAS procedures for GCNs.
- An update mechanism incorporating a knowledge reservoir to balance exploration and exploitation during the search process, leading to an accelerated and efficient search procedure while ensuring optimal exploration behavior.
- A diverse search space construction that enables the usage of different input representations to enhance the network performance and generalizability further.

II. RELATED WORK
A. SKELETON-BASED HAR
HAR within the domain of skeletal data attracts increasing interest due to its rectified robustness against background noise, occlusion, or scaling implications when compared to conventional RGB methods [3]. Since the nature of the skeleton data modality is non-euclidean, current approaches commonly use GCNs, as these can operate on unstructured data. The first work in this field stems from [7], introducing ST-GCN to learn spatial and temporal patterns from the input data with a tailored GCN layer. This foundation makes the handcrafted feature engineering obsolete while achieving superior performances compared to previous methods [7]. Several subsequent studies have adopted this technique and introduced multi-stream architectures to incorporate different kinematic variables from which the model can learn. A two-stream adaptive graph convolutional network named 2s-AGCN was introduced in [9] to consider the skeleton's second-order information and learn the graph's topology. In a later study [10], Efficient-GCN was introduced, built upon three input branches and incorporating techniques from separable convolution to make the network more efficient. While these approaches yield good performances, they are

all designed by human experts, limiting their generalizability to scenarios beyond the tested databases or usage by non-domain experts. This leaves the problem of the automatic construction of generic models unresolved.

B. NEURAL ARCHITECTURE SEARCH
For most tasks, deep neural networks must have a customized architecture to accomplish their objectives effectively. Initially, deep neural networks were used to automate the manual feature engineering aspect of machine learning. The field has since progressed further with the introduction of NAS and automated machine learning (AutoML), which automates the construction of neural architectures and accompanying tasks like hyperparameter optimization, e.g., batch size, learning rate, or algorithm selection [11]. These approaches have already given rise to architectures in the area of image classification that outperform models designed by human experts, as highlighted in studies such as [8] and [12].

Zoph et al. made significant contributions to the field of NAS with their pioneering work [8]. Their approach introduced a reinforcement learning algorithm searching for optimal architectures, yielding competitive performance on the CIFAR-10 and Penn Treebank benchmarks [13], [14]. However, it is worth noting that this method employed a substantial amount of computational resources, utilizing 800 GPUs over two weeks [12]. Significant research has focused on improving the efficiency of NAS approaches to address the first approach's resource-intensive aspect. For example, in the work that developed the Differentiable Architecture Search (DARTS) technique [15], a gradient-based optimization methodology was presented, enabling efficient exploration of designs by utilizing a continuous relaxation of the search space. Another influential study, [16], introduced the Efficient Neural Architecture Search (ENAS) approach, which addressed the computational inefficiency of NAS through parameter sharing across child models, yielding greater efficiency and scalability. The Progressive Neural Architecture Search (PNAS) [17] concept established a step-by-step expansion strategy to shorten the search process while maintaining competitive performance. Furthermore, the NASNet technique [18] focused on identifying transferable architectural building blocks that perform well across various tasks and datasets.

A new approach named AutoHAS incorporating hyperparameters and architecture components in the search was introduced in [19]. This comprehensive method aims to optimize the architectural design and the associated hyperparameters like the chosen optimizer and its learning rate for increased performance. Therefore, the shared network weights and a reinforcement learning controller are updated alternately to find the best probability distribution in the combined search space, resulting in the best-performing architecture and hyperparameter configuration. By extending the framework to incorporate hyperparameter optimization, AutoHAS went beyond the traditional scope of NAS and encompassed a broader spectrum of optimization possibilities. The inclusion

of hyperparameter search is a significant enhancement as it allows for the tuning of various factors that impact model performance, such as learning rates, regularization parameters, and batch sizes, which have a significant impact on the model performance [12]. In contrast, AutoHAS relies on the straightforward Reinforce algorithm [20], which overlooks previously sampled and trained candidate architectures during the update procedure, resulting in an imbalance between exploration and exploitation behaviors. Additionally, the architectural search is confined to basic building blocks, and the exploration of hyperparameters is restricted to selecting the most suitable learning rate and optimizer.

To date, NAS research has primarily focused on and applied to image classification tasks. In the domain of HAR GCNs, the NAS research focus has been limited to [21] and [22], to the best of our knowledge. In the work of [21], NAS was employed to discover the most effective architecture by approximating spatiotemporal cues and leveraging Chebyshev polynomials of varying orders. This approach involved constructing a search space consisting of three dynamic graph substructures: spatial, temporal, and spatiotemporal. Additionally, the authors searched for eight function modules that were subsequently applied to each network layer. Expanding upon these concepts, SNAS-GCN represents an extension of this research, optimizing the search space and implementing a single-path one-shot approach for improved search efficiency [22]. The experimental findings of SNAS-GCN demonstrate a reduction in search time compared to the previous work [21], albeit with a diminished level of accuracy. Despite the significant contributions of [21] and [22] to the field of skeleton GCNs through utilizing NAS, their work has certain limitations. One limitation lies in their reliance on approximating spatiotemporal cues and utilizing Chebyshev polynomials, which can lead to information loss and impact the derived architectures' accuracy and robustness. Furthermore, the search for a modest set of eight function modules applied to the network layers may restrict the expressiveness and flexibility of the discovered architectures. The lack of flexibility in the network architecture is a further notable limitation, as it is fixed to ten layers with uniform channel sizes and composition. This rigidity restricts the expressiveness of the architecture, potentially hindering its ability to capture complex patterns in the data effectively. In addition, the authors only search for the spatial and temporal filters solely on the joint data while sharing them with the other four dataset modalities within NTU RBG+D [23], and, as a result of this, overlook the potential variance present in these modalities. Moreover, the final performance of the architecture relies on an ensemble score derived from two distinct data modalities, making it necessary to use more resources to search and train those models. Considering these limitations, a more extensive and diverse search space combined with an ample data representation, the model could leverage a richer set of operations and adapt
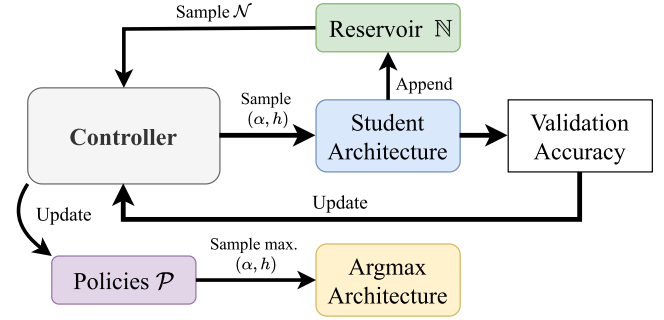

FIGURE 1. Overview of the proposed AutoGCN algorithm.

better to the specific task requirements, potentially leading to improved performance.

III. AUTOGCN

AutoGCN is a novel NAS framework that predicts action classes for a given sequence of skeletons. In this section, we provide AutoGCN's technical details. First, we introduce some preliminaries and the data representation as the foundational input features. Second, we discuss the defined search space and its implications. Third and finally, we explain the search algorithm in detail. Fig. (1) illustrates the workflow in the AutoGCN framework.

A. PRELIMINARIES
The human skeleton is modeled as a spatiotemporal graph by connecting the spatial graph of each time frame along a defined temporal dimension d, enabling the network to process spatial and temporal information effectively. Therefore, the skeleton sequence is transformed into an undirected graph described as $G = (V, E, \mathbf{A})$, where $n = |V|$ denotes the nodes connected by $|E|$ edges. These attributes are represented in an adjacency matrix $\mathbf{A} \in \mathcal{R}^{n \times n}$.

B. DATA REPRESENTATION
Prior research on skeleton-based action recognition demonstrated the paramount importance of data preprocessing [9], [24], [25], [26], [27], [28], [29]. In the context of this study, the input features are categorized into four distinct groups, namely: **1)** position P, **2)** velocity V, **3)** bone $B \in \{L, \beta\}$, and **4)** acceleration features A, corresponding to the fundamental kinematic properties of the human body [30]. The input sequence $\mathcal{X}$, denoted as $\mathcal{X} \in \mathcal{R}^{C_{in} \times T_{in} \times V_{in} \times M_{in}}$ represents a sequence of skeleton data inputs, where C_{in} denotes the number of channels indicating the $\{x, y, z\}$ orientation, T_{in} represents the number of frames in the sequence, V_{in} specifies the number of vertices representing spatial points, and M_{in} depicts the number of skeletons present in the sequence. This input sequence undergoes a subsequent partitioning into the mentioned categories. The relative position P is calculated via

$$\forall i = 1, 2, \ldots, V_{in}, \quad P = x[:, :, i, :] - x[:, :, c, :], \quad (1)$$

where c is the chosen central joint. The velocity V is determined via

$$\forall t = 1, 2, \ldots, T_{in}, \quad V = \frac{x[:,:,t+d,:] - x[:,:,t,:]}{d}. \tag{2}$$

The parameter d determines the timeframes over which the velocity is measured, with different values yielding different temporal spans. In this study, the range of d is within $[1, 2]$. To calculate the acceleration A, the velocity from (2) is differentiated with respect to time. Since the equation involves discrete time steps, finite differences are used to approximate the derivative. The acceleration A is then calculated as follows

$$\forall t = 1, 2, \ldots, T_{in}, \quad A = \frac{dV}{dt}, \tag{3}$$

$$\forall t = 1, 2, \ldots, T_{in} - d, \quad A = \frac{V[:,:,t+d,:] - V[:,:,t,:]}{d}. \tag{4}$$

An 8th-order Butterworth filter is also introduced to compensate by the derivative [31] from the acceleration feature. This meticulous filtering process is critical in enhancing the signal quality, enabling a more accurate and reliable representation of the underlying data. The bone features B are divided into bone length L and respective angles β which have been introduced in [10]:

$$\forall i = 1, 2, \ldots, V_{in}, \quad L = x[:,:,i,:] - x[:,:,i_{adj},:], \tag{5}$$

$$\forall j = 1, 2, \ldots, V_{in}, \quad \beta = cos^{-1}(\frac{l_{j,w}}{\sqrt{l_{j,x}^2 + l_{j,y}^2 + l_{j,z}^2}}). \tag{6}$$

Here i_{adj} is the adjacent joint, while $w \in \{x, y, z\}$ stands for the three orientation coordinates of the skeleton.

C. SPATIAL GRAPH CONVOLUTION
Yan et al. [7] pioneered the application of graph convolution in the domain of skeleton action recognition. The following relation encapsulates the essence of their approach:

$$f_{out}(v_{ti}) = \sum_{v_{tj} \in B(v_{ti})} \frac{1}{Z_{ti}(v_{tj})} f_{in}(v_{tj}) \mathbf{w}(l_{ti}(v_{tj})). \tag{7}$$

Here, v_{ti} represents the i-th joint at the t-th frame, and $f_{out}(.)$ and $f_{in}(.)$ correspond to the output and input features of the respective joints. Furthermore, $B(v_{ti})$ denotes the set of neighboring joints of v_{ti}, while $B_{v_{ti}}$ defines the convolutional sampling area, specifically encompassing vertexes that are 1-distance neighbors (i.e., immediate neighboring joints). The normalization factor $Z_{ti}(v_{tj})$ represents the size of the corresponding subset, and its purpose is to equalize the influence of various subsets on the final output. The weighting function $\mathbf{w}(.)$ incorporates the mapping function $l_{ti}(.)$, which serves to establish diverse partitioning strategies, such as uni-labeling, distance, and spatial configuration partitioning [7]. To further

refine the formulation, Equation (7) is written as

$$\mathbf{f}_{out} = \sum_{j} \mathbf{W}_j \mathbf{f}_{in}(\Lambda_j^{-\frac{1}{2}} \mathbf{A}_j \Lambda_j^{-\frac{1}{2}} \mathbf{M}_j). \tag{8}$$

Here, $\mathbf{f}_{out}$ denotes the output features, while $\mathbf{W}_j, \Lambda_j, \mathbf{A}_j$, and $\mathbf{M}_j$ contribute to the weighting, diagonal degree, adjacency, and normalization matrices, respectively. This transformation enables an enhanced representation of the skeleton sequences using graph convolution techniques [7].

D. SEARCH SPACE
Diverging from conventional image-based NAS algorithms, which typically incorporate a limited set of elementary building blocks, such as convolutional layers with varying kernel sizes, skip connections, or max pooling layers [32], [33], AutoGCN adopts a distinctive approach. Recognizing that these simplistic blocks cannot adequately capture the data representation of a skeletal sequence, AutoGCN embraces the integration of prior knowledge. This algorithm aims to exploit the full potential of skeletal sequence information by defining the search space using architectural components known to be suitable for skeletal data [7], [9], [16], [21], [24], [34]. This departure from traditional methodologies acknowledges the need for a more nuanced and informed exploration of possible architectures in this domain [12], [15]. In addition to incorporating prior knowledge in the search space, AutoGCN also encompasses hyperparameters like optimizer type, momentum, batch size, L2 regularisation, and learning rate. Building upon [19], the hyperparameter search space can be formulated as

$$h = \sum_{i=1}^{m} C_i^h \mathcal{B}_i \ s.t. \ \sum_{i=1}^{m} C_i^h = 1,$$

$$\text{and } C_i^h \in \begin{cases} [0, 1], & \text{if cont.} \\ \{0, 1\}, & \text{if categ.,} \end{cases} \tag{9}$$

where the hyperparameter h is a sum of weighted components C_i^h of the predefined values $\mathcal{B}_i$, satisfying two conditions: the weights sum up to 1, to ensure these are normalized and that the combined influence of all components adds up to the final hyperparameter value, and C_i^h behaves either as a continuous value in $[0, 1]$ or as a binary choice $\{0,1\}$ depending on whether it represents a continuous or categorical hyperparameter.

The architectural search space can be encapsulated in the following equation [19]

$$\alpha = \sum_{i=1}^{n} \sum_{j=1}^{k} C_{i,j}^{\alpha} \Delta_{i,j} \ s.t. \ \sum_{i=1}^{n} \sum_{j=1}^{k} C_{i,j}^{\alpha} = 1,$$

$$\text{and } C_{i,j}^{\alpha} \in \begin{cases} [0, 1], & \text{if cont.} \\ \{0, 1\}, & \text{if categ.,} \end{cases} \tag{10}$$

where n defines the chosen architecture type $C_{i,j}^{\alpha}$ and k the corresponding value out of the predefined set of architectural parameters $\Delta_{i,j}$. Just as with the hyperparameters, the
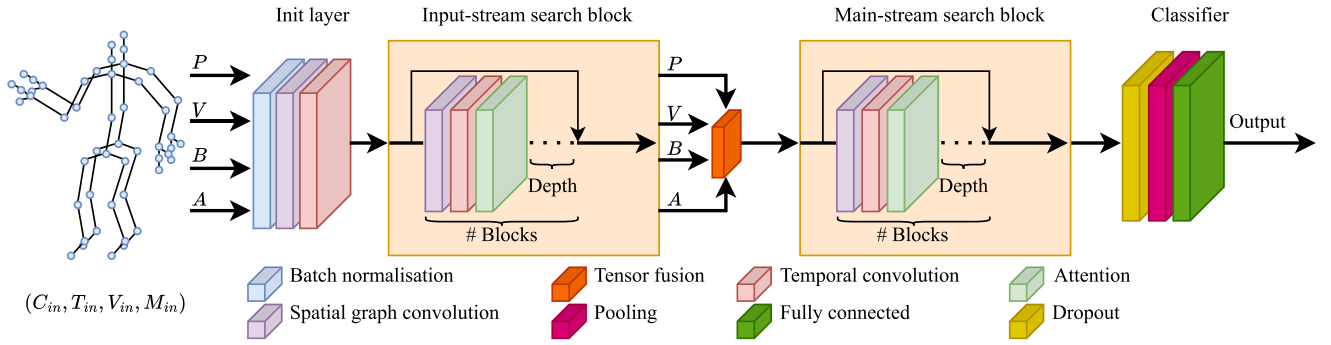
FIGURE 2. Architecture skeleton of the AutoGCN algorithm.

TABLE 1. Search Space comprising the parameter and the corresponding value ranges, grouped into the effective areas.

Parameter	Range
Common	
Activation layer	[Relu, Relu6, Hardswish, Swish]
Attention layer	[Stja, Ca, Fa, Ja, Pa] [10]
Conv. layer type	[Basic, Bottleneck[10], Sep[10] SG[10], V3[35], Shuffle[36]]
Dropout probability	[0.15, 0.2, 0.25, 0.3]
Init layer size	[64, 128, 156, 195, 256]
Input stream	
Blocks in	[1, 2, 3]
Depth in	[1, 2, 3]
Stride in	[1, 2, 3]
Scaling factor	[0.4, 0.6, 0.8]
Temporal window	[3, 5, 7]
Graph distance	[2, 3, 4]
Reduction factor	[1.225, 1.25, 1.275, 1.3, 1.325, 1.35]
Main stream	
Blocks main	[3, 4, 5, 6]
Depth main	[1, 2, 3, 4]
Graph distance	[7, 9, 11]
Shrinkage	[1, 2, 4, 6]
Residual layer	[True, False]
Adaptive	[True, False]
Optimizer	
Optimizer	[SGD, Adam, AdamW]
Learning rate	[0.1, 0.05, 0.01]
Weight decay	[0.0, 0.01, 0.001, 0.0001]
Momentum	[0.5, 0.9, 0.99]
Batch size	[8, 16, 24]

architecture can be categorical or continuous. By considering both the architectural components and the hyperparameters, AutoGCN ensures a comprehensive search process, allowing for the fine-tuning of multiple parts to achieve an optimal outcome. Table (1) shows the entire search space and its variables.

E. ARCHITECTURE

As previously discussed, the architecture construction necessitates careful consideration of the unique data representation intrinsic to the skeleton sequence. To address this requirement effectively, the architecture is thoughtfully partitioned into separate building blocks, building upon prior research [10], [29], [37]. The fundamental elements comprise **1)** the initialization layer, **2)** the input stream- and, **3)** the main stream search block, followed by **4)** the classifier block, as depicted in Fig. (2). Each block integrates various parameters derived from the comprehensive search space in Table (1). This modular approach ensures that the architecture can deal with the intricate nuances and complexities inherent in the skeleton sequence data representation.

F. SEARCH ALGORITHM

AutoGCN seeks optimization on two fronts: the hyperparameters denoted as h, and the architecture represented as α. This approach aligns with prior research in this field [12], [19]. The overarching objective is to minimize the loss function $\mathcal{L}$ on the validation dataset $\mathcal{D}_{val}$ while ensuring that the optimal parameters $\omega^*_{\alpha,h}$ are derived from the minimized loss on both the architecture α, hyperparameters h and the training dataset $\mathcal{D}_{train}$. This is articulated as follows:

$$\min_{\alpha,h} \mathcal{L}(\alpha, \omega^*_{\alpha,h}, \mathcal{D}_{val}) \quad \text{s.t.} \quad \omega^*_{\alpha,h} = \mathcal{L}(\alpha, h, \mathcal{D}_{train}). \quad (11)$$

To address the potential inefficiencies of the Reinforce algorithm's sampling behavior, AutoGCN incorporates a replay memory $\mathbb{N}$ that stores a collection of $\mathcal{N}$ previously generated student architectures. This memory is used to enhance the sampling procedure for updating the controller. The addition of a replay memory helps mitigate the limitations of the Reinforce algorithm by utilizing a broader range of student architectures that have been saved over time. This has the effect of expanding the effective sampling size and improving the controller's decision-making process. Refer to Algorithm (1) for further details.

IV. EXPERIMENTS

This section outlines our experimental setup of the proposed AutoGCN algorithm on two datasets: NTU RGB+D 60 and NTU RGB+D 120 [23], [38]. We compare current state-of-the-art (SOTA) models and the two introduced NAS procedures on these datasets. Furthermore, we investigate the impact of controller hyperparameters on the model's performance, including the number of rollouts and update

---

Algorithm 1 AutoGCN Algorithm

Input: Split the data into: $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$
Output: $\mathcal{P}_{max}(\alpha, h)$
Initialize controller's policies $\mathcal{P}$, iterations i, and
   rollouts r
while *not converged* **do**
   for i **do**
      Sample (α, h) from controller's search space
      Build student architecture
      for 25 epochs **do**
         Train student architecture
         Append validation accuracy to r
   Sample $\mathcal{N}$ student architectures from reservoir $\mathbb{N}$
   Update controller by REINFORCE with r
   Sample $\mathcal{P}_{max}(\alpha, h)$ and train architecture
Get final architecture from $\mathcal{P}_{max}(\alpha, h)$

---

cycles. Additionally, we perform experiments to assess the importance of the acceleration feature in our data representation and the influence of the searched network size.

A. DATASETS
NTU RGB+D 60 is a substantial 3D human activity dataset for action recognition comprising two versions: NTU RGB+D 60 [23] and NTU RGB+D 120 [38]. NTU RGB+D 60 contains 56,880 videos, each representing one of the 60 action classes. NTU RGB+D 120 is an extension of NTU RGB+D 60, including 60 additional action classes and 114,480 videos. To evaluate the classification performance, we follow the cross-subject and cross-view settings for NTU RGB+D 60 [23], and cross-subject and cross-setup settings for NTU RGB+D 120, as suggested respectively in [23] and [38].

The NTU RGB+D 60 dataset is partitioned and used for evaluation as follows: (1) Cross-Subject (X-Sub) evaluation divides the dataset based on subjects. Specifically, 20 subjects are assigned to the training set, while the remaining 20 form the test set. (2) Cross-View (X-View) evaluation involves partitioning the dataset based on camera views. For this evaluation, camera views two and three are utilized to create the training data, while camera view one is reserved for testing.

Correspondingly, the following split protocols are suggested for NTU RGB+D 120 by the authors: (1) Cross-Subject (X-Sub120): The training set comprises samples from 56 subjects, while the test set includes samples from 50 subjects. (2) Cross-Setup (X-Setup120): In this protocol, samples with even setup IDs are designated for training, while samples with odd setup IDs are reserved for testing.

B. IMPLEMENTATION
In the experimental setup, the student architecture is trained for a maximum of 25 epochs, while the argmax architecture is trained for 80 epochs to ensure complete convergence. The learning rate used in training is sampled from the search space and undergoes a warm-up for the first ten epochs with a decay factor of 0.5. Subsequently, the learning rate is reduced by 0.25 at epochs 30, 50, 60, 65, and 70. An early stopping mechanism is employed to identify underperforming student architectures during the initial training phase, terminating the training process before the sixth epoch if the accuracy is below 50%.

The controller is updated with the Adam optimizer and a learning rate of 0.001. Once the predetermined number of student architectures have been sampled and trained, the controller is updated with those achieved accuracies as rewards. Simultaneously, sampling from the replay memory continues, and only student architectures achieving an accuracy higher than 80% during the training process are added to the replay memory.

The training is performed on a single NVIDIA-V100 with 32 GB GPU RAM on the PyTorch framework (version 2.0.1) [39] with the global seed set to 1234. The code and the experiment results are publicly available at https://github.com/DeepInMotion/AutoGCN.

C. COMPARISON WITH OTHER SOTA
To evaluate the performance of AutoGCN, it is compared with the baseline approach from Dong et al. [19] and other state-of-the-art (SOTA) HAR approaches. The best-performing models' results on the NTU RGB+D 60 database are shown in Table (2) and (3), respectively. The values of $\mathcal{P}_{max}(\alpha, h)$ for the best-performing model are listed in Table (4). Moreover, the values of the Policies $\mathcal{P}$ for all search space parameters are listed in Appendix (A) Fig. (4) and Appendix (B) Fig. (5), accompanied by an analysis of these values.

In contrast to other SOTA methodologies, we present the achieved point estimate for the NTU RGB+D 60 dataset and the associated confidence interval, calculated using bootstrap resampling [40]. Specifically, we perform 1000 resamples of the complete test dataset with replacement while maintaining a constant training set and model configuration. Our reporting is based on the boundaries encompassing the [2.5, 97.5] percentiles, establishing a 95% confidence interval for the test set.

As shown in Table (2), our approach surpasses the NAS-GCN baseline with the Joint configuration by 0.8% for the X-View and the X-Sub dataset. In contrast to SNAS-GCN, our model demonstrates a notable increase in accuracy, with a substantial gain of 1.2% observed for both the X-Sub and X-View datasets. Moreover, it is essential to emphasize that AutoGCN achieves similar results to the baseline method NAS-GCN without the need for ensemble techniques, achieving a competitive accuracy of 95.5% on the X-View dataset. These results show the effectiveness and efficiency of AutoGCN in the context of skeleton-based human action recognition by delivering high performance without the added complexity of ensemble methods.

TABLE 2. Comparison with SOTA models and NAS approaches on the NTU RGB+D 60 dataset with the Top-1 accuracy in (%). The square brackets indicate the 95% confidence intervals, determined via bootstrapping.

Model	X-Sub	X-View
ST-GCN [7]	81.5	88.3
AS-GCN [41]	86.8	94.2
GR-GCN [42]	87.5	94.3
2S-AGCN [9]	88.5[1]	95.1[1]
AGC-LSTM [43]	89.2	95.0
SGN [29]	89.0	94.5
MST-AGCN [44]	89.5[1]	95.5[1]
EfficientGCN-B0 [10]	90.2	94.9
STGAT (Joint) [34]	90.2	95.4
SNAS-GCN [22]	89.0[1]	95.0[1]
SNAS-GCN (Joint) [22]	87.1	94.3
NAS-GCN [21]	89.4[1]	95.7[1]
NAS-GCN (Joint) [21]	87.5	94.7
AutoGCN	88.3 [86.9, 89.2]	95.5 [94.6, 96.3]

TABLE 3. Comparison with SOTA models on the NTU RGB+D 120 dataset with the Top-1 accuracy in (%).

Model	X-Sub120	X-Setup120
ST-GCN [7]	70.7	73.2
AS-GCN [41]	77.9	78.5
2S-AGCN [9]	82.5[1]	84.2[1]
SGN [29]	79.2	81.5
EfficientGCN-B0 [10]	86.6	85.0
AutoGCN	83.3	84.1

Table (3) demonstrates that our approach yields results comparable to the SOTA models for the NTU RGB+D 120 dataset previously reported.

As delineated in Table (4), a discernible disparity in the identified architectural components between the X-Sub and X-View datasets emerges. This disparity supports our contention from the introduction that these two datasets have inherent characteristics that cannot be effectively captured by a single searched architectural framework alone.

D. COMPARISON TO RANDOM SEARCH

Random search [45] as an optimization technique for tuning hyperparameters is widely used in NAS procedures, which makes it an essential component of our experiments [46]. To ensure the fairest comparison against our proposed method, we perform the random search following the same steps as our algorithm AutoGCN: An initial cohort of 20 student architectures are trained for 25 epochs, and subsequently, the highest-performing architecture is chosen and trained for an extended training period of 80 epochs. Finally, we compare the iterations taken by each approach to obtain the best-performing architectures, displayed in Table (5). It can be observed that the AutoGCN algorithm achieves a higher-performing architecture after just 20 iterations both for the X-View and X-Sub datasets compared to random search. To achieve a comparable point-estimates as AutoGCN, random search has to undergo

[1]ensemble score

TABLE 4. Comparison of the found architecture and hyperparameter components from the best performing models.

Parameter	X-Sub	X-View
Common		
Activation layer	Hardswish	Hardswish
Attention layer	Fa	Ja
Conv. layer type	Bottleneck	SG
Dropout probability	0.25	0.15
Init layer size	64	256
Input stream		
Blocks in	1	2
Depth in	2	1
Stride in	3	3
Scaling factor	0.4	0.6
Temporal window	5	3
Graph distance	3	2
Reduction factor	1.25	1.35
Main stream		
Blocks main	3	3
Depth main	1	3
Graph distance	11	9
Shrinkage	2	2
Residual layer	True	True
Adaptive	True	True
Optimizer		
Optimizer	SGD	AdamW
Learning rate	0.05	0.01
Weight decay	0.001	0.0
Momentum	0.5	0.5
Batch size	16	8

TABLE 5. Comparison of random search and AutoGCN to the period of time and the Top-1 accuracy in (%).

	X-Sub	X-View
Random search	86.8 - 20 iterations	92.9 - 20 iterations
AutoGCN	**88.3 - 20** iterations	**95.1 - 20** iterations

another 20 iterations. The results indicate that AutoGCN can obtain an optimal architectural configuration without being subject to the stochasticity of random search.

E. INFLUENCE OF THE CONTROLLER HYPERPARMETERS

Since the controller has fixed hyperparameters, we investigate the ideal number of rollouts for training our model, aiming to balance computational efficiency and the final model performance. The found values are shown in Table (6), in which the Top-1 accuracy percentages in relation to the rollout hyperparameter of the controller and the update stages are depicted. It also displays the average accuracies of the student models and Top-1 accuracies for the first, second, and third updates in percentage values. The table provides an overview of how the choice of rollouts impacts the accuracy of the two datasets, X-Sub and X-View. The arrow symbols describe the trend in the accuracy and average accuracy compared to the previous update: '↑' for an increase, '↓' for a decrease, '→' for no significant change. Furthermore, those results are visualized in Fig. (3).
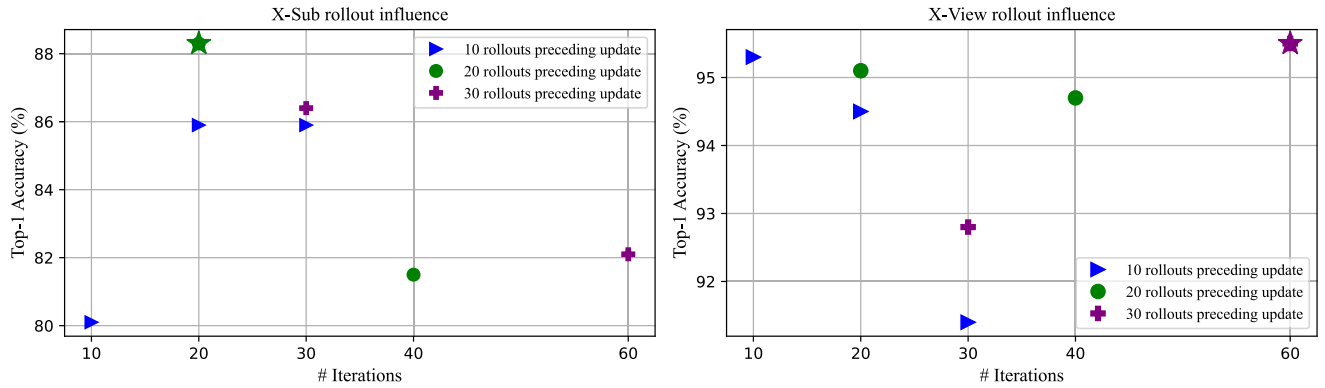
FIGURE 3. Influence of the rollout parameter on the model performance. The star indicates the highest accuracy achieved in the rollout experiments.

TABLE 6. Top-1 accuracy percentages w.r.t the rollout hyperparameter of the controller and the update states, along with the average accuracies of the student models and Argmax accuracies for the update cycles in (%). The table showcases how the choice of rollouts impacts the accuracy of two distinct approaches, X-Sub and X-View, and the effects of the first, second, and third updates on their respective performance.

Rollouts	Top-1	1st update Avg. - Top-1	2nd update Avg. - Top-1	3rd update Avg. - Top-1
X-Sub				
10	85.9	50.4 - 80.1	61.3 ↑ - **85.9** ↑	62.3 ↑ - 85.9 →
20	**88.3**	45.7 - **88.3**	33.5 ↓ - 81.5 ↓	–
30	86.4	49.6 - **86.4**	60.3 ↑ - 82.1 ↓	–
X-View				
10	95.3	71.3 - **95.3**	72.0 → - 94.5 ↓	68.6 ↓ - 91.4 ↓
20	95.1	56.2 - **95.1**	66.4 ↓ - 94.7 →	–
30	**95.5**	69.1 - 92.8	60.0 ↓ - **95.5** ↑	–

The X-View dataset attains its highest point estimate following 30 rollouts after the second update cycle. Notably, for 30 rollouts, the controller identifies the best-performing architecture after the second update, requiring significantly more time than the experiment with only ten or 20 samples between updates while not achieving a notable higher point estimate.

Conversely, the X-Sub dataset achieves its highest point estimate after 20 rollouts, which also transpires following the first update cycle. The X-Sub dataset necessitates only one update cycle to achieve the highest point estimate for 20 and 30 rollouts. With ten rollouts, the controller achieves the highest accuracy after the second iteration and remains constant afterward. With 30 rollouts, the controller reaches a state where it becomes more confident about a suboptimal model configuration, resulting in diminished performance.

Given that the student architectures are trained for a shorter duration of 25 epochs, the optimizer's hyperparameters significantly impact the final accuracy used as a reward for the controller. With a "fast learning" optimizer, suboptimal student architectures, which converge more rapidly, may enjoy an advantage over "slower" trained architectures in the early stages of training since those are only trained for 25 epochs.

TABLE 7. Changed values of the search space and the resulting point estimate in (%) on the X-View dataset.

Search Space			
Init layer size	[156, 195, 256, 320]		
Blocks in	[2, 3, 4]		
Depth in	[3, 4, 5]		
Blocks main	[5, 6, 7, 8]		
Depth main	[3, 4, 5]		
Performance	FLOPs	Parameters	Top-1
Large model	13.94G	4.79M	95.4
Small model	14.47G	1.76M	95.3

Considering that the selection of the student architecture is stochastic, the controller may be updated with less performant models between each update cycle. This becomes evident when comparing the average accuracy for the X-Sub dataset with ten rollouts, where a notable rise in the average accuracy correlates with an increase in the highest point estimate. On the other hand, the second update with 30 rollouts decreases the point estimate for the X-Sub datasets, suggesting that the controller becomes trapped in a suboptimal configuration after this amount of updates. Furthermore, it has to be recognized that the search space contains multiple optima that can be explored, resulting in the alteration of optimal architecture builds. Consequently, different architecture configurations can achieve similar performances after optimization, which becomes evident when investigating the different sampled architecture configurations across the sampled rollouts.

F. INFLUENCE OF THE MODEL SIZE
We conduct experiments to investigate the optimal network size by altering the search space, as outlined in Table (1). These focus on exploring larger architectures, where we increased layers, blocks, and depth sizes in the experimental design. The results in Table (7) indicate that the X-View dataset's accuracy remains consistent for the larger and smaller sampled architecture after 20 rollouts. The small model requires much fewer parameters but slightly more FLOPs than the larger model. The greater amount of FLOPs

TABLE 8. Influence of the acceleration feature *A* on the Top-1 accuracy in (%) with the 95% confidence interval.

Input	X-Sub	X-View
P, V, B	87.5 ± 1.4	93.7 ± 1.1
P, V, B, A	88.3 ± 0.9	95.6 ± 0.8

is due to the more complex *Bottleneck* convolutional layer chosen by the controller.

This outcome shows the effectiveness of the proposed search space, as defined in Table (1), and emphasizes the usage of smaller models that require significantly less training. The results advocate utilizing these smaller models for their computational efficiency while achieving the same performance as the large model, requiring fewer computational resources.

G. DATA REPRESENTATION

The newly introduced acceleration feature is analyzed to gauge the strength of the data representation. Consequently, this feature is excluded and incorporated into the highest-performing model identified through our search. The impact of omitting this feature from the model becomes evident when observing the results presented in Table (8), where the Top-1 accuracy demonstrates a notable decrease. When conducting a two-sided z-test with a 5% significance level, the null hypothesis of no significant difference between those data modalities can be rejected.

V. LIMITATIONS AND FUTURE WORK

The proposed AutoGCN algorithm uses a reinforcement learning approach, requiring every student network to be trained from scratch at each iteration. While this strategy enables dynamic adaptation and refinement of the individual architecture and hyperparameter components, it also leads to an increased computational burden in computing the reward for the controller guiding the search and updating the policies $\mathcal{P}$.

In order to avoid having to train every student architecture from scratch, the search procedure could adopt so-called *one-shot* techniques like DARTs, in which an over-parameterized *supernetwork* is trained to contain every potential architecture build and enable weight sharing among students [12], [15]. To implement this approach, reusable building blocks, and operation types that can be recycled among different student compositions effectively would have to be defined. While such an adaptation could then lead to a speed-up of the search procedure, it could also lower the versatility of the search space.

In our work, speed-up techniques for the search process are limited to leveraging the knowledge reservoir $\mathcal{N}$ and implementing an early-stopping mechanism. Possible enhancements to accelerate the search could involve the inclusion of performance predictors [12]. Such predictors could assess the potential of a given student architecture early, allowing for a more efficient allocation of computational resources

by prioritizing the exploration of promising candidates and neglecting unpromising ones. With such techniques, the convergence towards optimal solutions could be accelerated, resulting in a significantly reduced overall search time.

VI. CONCLUSION

In this study, we have developed and presented a GCN NAS algorithm named AutoGCN, tailored for the task of skeleton-based HAR. The intricate dependencies between hyperparameters and architectural configurations are formulated in an expressive search space encompassing a broad range of building blocks from which the controller can sample, which allows the algorithm to find a versatile and high-performing network architecture. AutoGCN is applied to identify and optimize these fundamental building blocks of the network concurrently with a reinforcement algorithm, giving every search space parameter a policy that can be updated based on the anticipated performance from the sampled student architectures. The search process's exploration and exploitation behavior is refined by incorporating a replay memory during the search process, enabling the method to strike a promising balance between these and enhancing the overall search performance.

Through extensive experiments, we provide a rigorous performance analysis, comparing our method against the baseline NAS procedure and SOTA approaches in the domain of skeleton-based HAR. Finally, AutoGCN demonstrates effectiveness also compared to random search.

Future work will investigate the influence of performance predictors on AutoGCN to decrease the search time of the algorithm, as good-performing architectures could be recognized earlier. Additionally, weight-sharing techniques could further make the search more efficient.

APPENDIX A POLICY VALUES ON THE X-SUB DATASET

Fig. (4) shows the policy values for the model's hyperparameters and architecture search space with 20 rollouts between controller updates on the X-Sub dataset. These values are categorized within the four search areas as shown in Table (1). The highest Top-1 accuracy in this experiment is attained with the first controller update, with the final $\mathcal{P}_{max}(\alpha, h)$ values outlined in Table (4).

Within the *Common* group, most policy values maintain consistency after the second update cycle. However, within the *Conv. layer* choices, uncertainty arises, particularly evident in the fluctuation of probabilities between the *Basic* and *Shuffle* layers after the first controller update. A similar trend of uncertainty is visible in the *Dropout probability* and *Init layer size*. Regarding the *Input stream* group, discernible trends emerge after both the first and second update cycles. The search space parameters with lower probability values tend to retain their trend after the first update. Nonetheless, the controller is uncertain regarding *Blocks in*, *Stride in*, and the *Reduction factor*, with a tight distribution across these search space parameters for the highest values. In the *Main stream* group, most search areas demonstrate clear
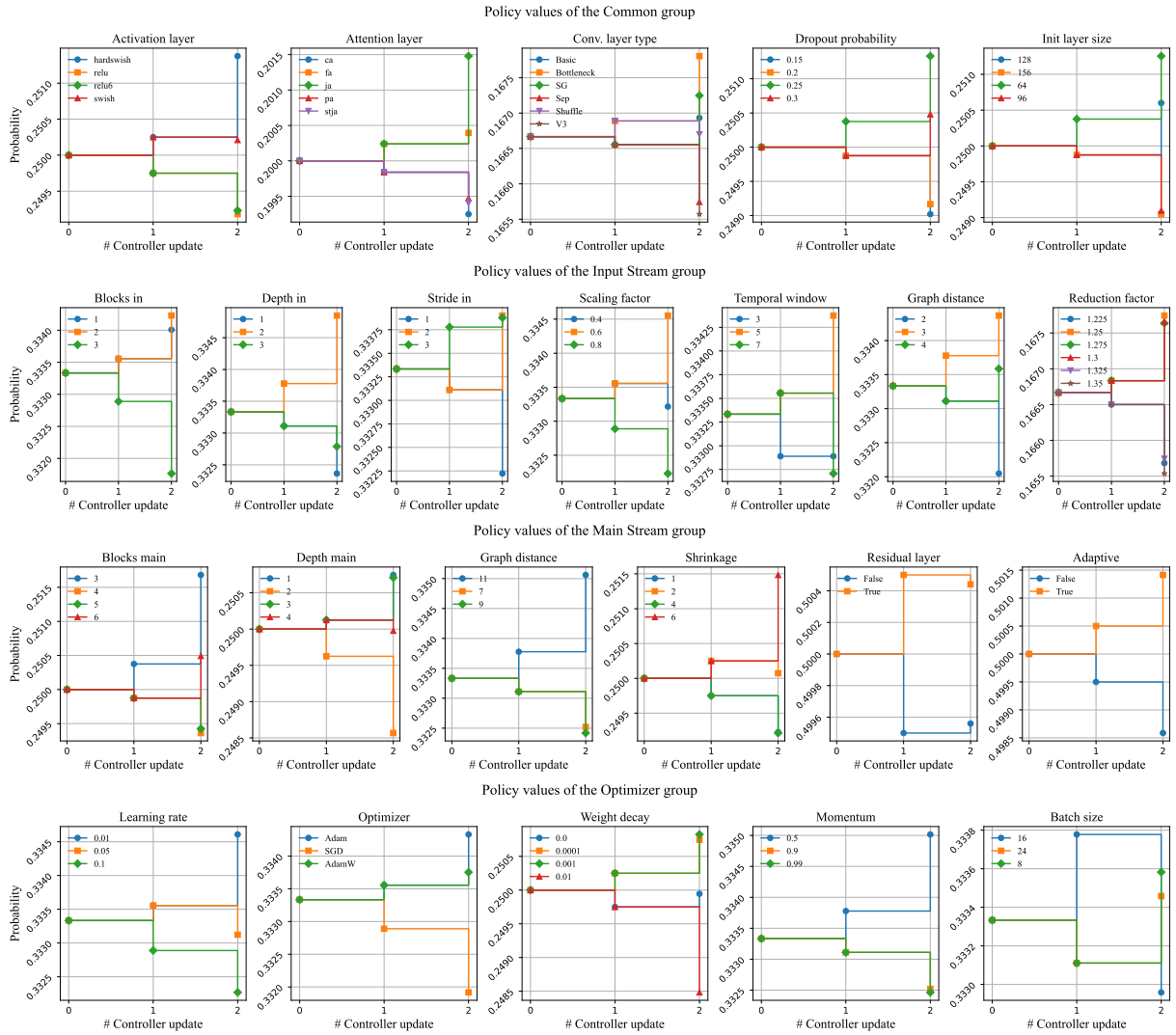
FIGURE 4. Policy values for the best-performing model on the X-Sub dataset. One update cycle contains 20 student architectures.

trends, barring the values associated with *Depth main*. These probability values for the depths of one or three layers remain closely clustered following the first and second controller updates. Within the *Optimizer* group, variability among the policy values is notable, particularly for the *Learning rate* and *Batch size* parameters. Given the intertwined nature of these hyperparameters and their impact on the incompletely trained student architecture, fluctuations are expected.

It is essential to highlight that multiple local optima are possible within this search space configuration. These optima depend upon the statistical variance from the random sampling process of the student architectures. Furthermore, the average student's Top-1 accuracy is significantly lower with the second update cycle in this experiment compared to the other rollout experiments from Table (6).

APPENDIX B POLICY VALUES ON THE X-VIEW DATASET
Fig. (5) shows the policy values for the model's hyperparameter and architecture search space with 30 rollouts between

controller updates on the X-View dataset. The values are grouped as defined in Table (1). The best Top-1 accuracy in this experiment is achieved with the second controller update, where the $\mathcal{P}_{max}(\alpha, h)$ values are presented in Table (4).

For the *Common* group, the trend from the first to the second controller update is relatively consistent in its upward direction for the highest search space value. In contrast, the final value for the *Init layer size* is experiencing a significant increase in its probability value. The policy values with lower probabilities fluctuate more between the first and second controller updates. In the *Input stream* group, most policy values indicate a clear trend for the one single search space value. In contrast, the *Graph distance* and the *Reduction factor* show variability between the first and the second update. Regarding the *Main stream* group, most search areas manifest clear trends, except for the values associated with *Depth main*. As in the X-Sub experiment, the probability values for depths of one or three layers remain closely clustered following the first and second controller updates.
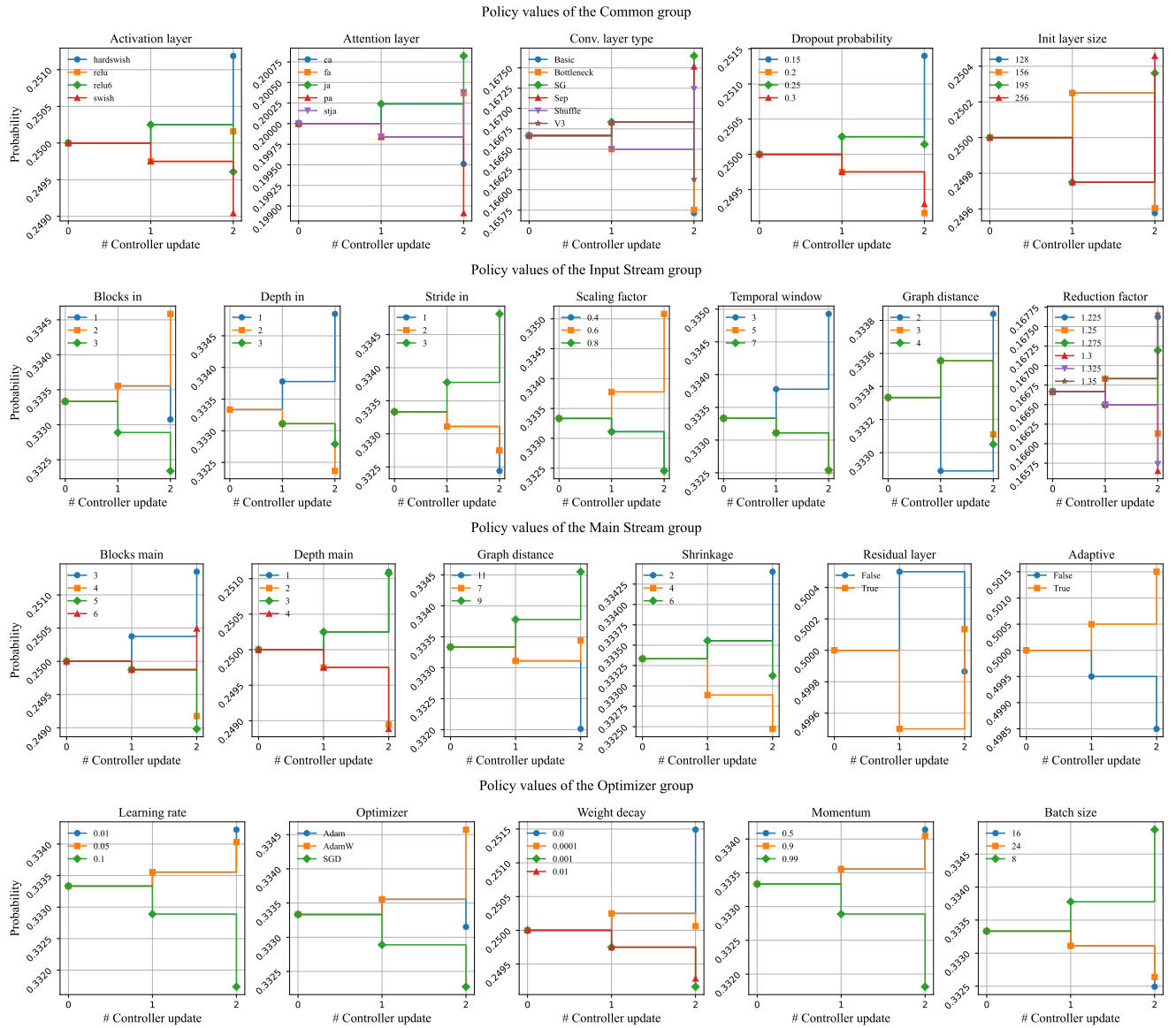
FIGURE 5. Policy values for the best-performing model on the X-View dataset. One update cycle contains 30 student architectures.

Additionally, the values for the *Residual layer* converse after the second update. In the *Optimizer* group, the policy values indicate less fluctuation than for the X-Sub dataset in Fig. (4). A clear trend among these different groups is visible from the first to the second controller update.

As with the X-Sub experiments, the search space values contain a statistical variance due to the random sampling process of the student architectures.

REFERENCES

[1] M. Feng and J. Meunier, "Skeleton graph-neural-network-based human action recognition: A survey," *Sensors*, vol. 22, no. 6, p. 2091, Mar. 2022, doi: 10.3390/s22062091.

[2] D. Groos, "Development and validation of a deep learning method to predict cerebral palsy from spontaneous movements in infants at high risk," *JAMA Netw. Open*, vol. 5, no. 7, Jul. 2022, Art. no. e2221325, doi: 10.1001/jamanetworkopen.2022.21325.

[3] M. Vrigkas, C. Nikou, and I. A. Kakadiaris, "A review of human activity recognition methods," *Frontiers Robot. AI*, vol. 2, p. 28, Nov. 2015, doi: 10.3389/frobt.2015.00028.

[4] M. H. Arshad, M. Bilal, and A. Gani, "Human activity recognition: Review, taxonomy and open challenges," *Sensors*, vol. 22, no. 17, p. 6463, Apr. 2022, doi: 10.3390/s22176463.

[5] X. Yang and Y. L. Tian, "EigenJoints-based action recognition using Naive-Bayes-nearest-neighbor," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops*, RI, USA, Jun. 2012, pp. 14–19, doi: 10.1109/CVPRW.2012.6239232.

[6] A. F. Bobick and J. W. Davis, "The recognition of human movement using temporal templates," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 3, pp. 257–267, Mar. 2001, doi: 10.1109/34.910878.

[7] M. Jiang, J. Dong, D. Ma, J. Sun, J. He, and L. Lang, "Inception spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. Int. Symp. Control Eng. Robot. (ISCER)*, Feb. 2022, pp. 208–213.

[8] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017, *arXiv:1611.01578*.

[9] L. Shi, Y. Zhang, J. Cheng, and H. Lu, "Two-stream adaptive graph convolutional networks for skeleton-based action recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12018–12027.

[10] Y.-F. Song, Z. Zhang, C. Shan, and L. Wang, "Constructing stronger and faster baselines for skeleton-based action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 2, pp. 1474–1488, Feb. 2023.

[11] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges* (Springer Series on Challenges in Machine Learning). Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-05318-5.

[12] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, and F. Hutter, "Neural architecture search: Insights from 1000 papers," 2023, *arXiv:2301.08727*.

[13] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[14] M. P. Marcus and M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Comput. Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[15] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. ICLR*, Apr. 2019, pp. 1–27.

[16] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018, *arXiv:1802.03268*.

[17] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," 2018, *arXiv:1712.00559*.

[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," 2018, *arXiv:1707.07012*.

[19] X. Dong, M. Tan, A. W. Yu, D. Peng, B. Gabrys, and Q. V. Le, "AutoHAS: Efficient hyperparameter and architecture search," 2021, *arXiv:2006.03656*.

[20] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 1999, pp. 1–12.

[21] W. Peng, X. Hong, H. Chen, and G. Zhao, "Learning graph convolutional network for skeleton-based human action recognition by neural searching," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 2669–2676.

[22] Y. Jiang, S. Yu, T. Wang, Z. Sun, and S. Wang, "Skeleton-based human action recognition based on single path one-shot neural architecture search," *Electronics*, vol. 12, no. 14, p. 3156, Jul. 2023, doi: 10.3390/electronics12143156.

[23] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "NTU RGB+D: A large scale dataset for 3D human activity analysis," 2016, *arXiv:1604.02808*.

[24] Y. Chen, Z. Zhang, C. Yuan, B. Li, Y. Deng, and W. Hu, "Channel-wise topology refinement graph convolution for skeleton-based action recognition," 2021, *arXiv:2107.12213*.

[25] R. Li, H. Fu, W.-L. Lo, Z. Chi, Z. Song, and D. Wen, "Skeleton-based action recognition with key-segment descriptor and temporal step matrix model," *IEEE Access*, vol. 7, pp. 169782–169795, 2019, doi: 10.1109/ACCESS.2019.2954744.

[26] Q. Nie, J. Wang, X. Wang, and Y. Liu, "View-invariant human action recognition based on a 3D bio-constrained skeleton model," *IEEE Trans. Image Process.*, vol. 28, no. 8, pp. 3959–3972, Aug. 2019, doi: 10.1109/TIP.2019.2907048.

[27] X. Qin, H. Li, Y. Liu, J. Yu, C. He, and X. Zhang, "Multi-stage part-aware graph convolutional network for skeleton-based action recognition," *IET Image Process.*, vol. 16, no. 8, pp. 2063–2074, Jun. 2022, doi: 10.1049/ipr2.12469.

[28] P. Zhang, C. Lan, W. Zeng, J. Xing, J. Xue, and N. Zheng, "Multi-scale semantics-guided neural networks for efficient skeleton-based human action recognition," 2021, *arXiv:2111.03993*.

[29] P. Zhang, C. Lan, W. Zeng, J. Xing, J. Xue, and N. Zheng, "Semantics-guided neural networks for efficient skeleton-based human action recognition," 2020, *arXiv:1904.01189*.

[30] V. M. Zatsiorsky, *Kinematics Human Motion*. Champaign, IL, USA: Human Kinetics, 1998.

[31] F. Crenna, G. B. Rossi, and M. Berardengo, "Filtering biomechanical signals in movement analysis," *Sensors*, vol. 21, no. 13, p. 4580, Jul. 2021, doi: 10.3390/s21134580.

[32] X. Dong and Y. Yang, "NAS-bench-201: Extending the scope of reproducible neural architecture search," 2020, *arXiv:2001.00326*.

[33] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.

[34] L. Hu, S. Liu, and W. Feng, "Spatial temporal graph attention network for skeleton-based action recognition," 2022, *arXiv:2208.08599*.

[35] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," 2019, *arXiv:1905.02244*

[36] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," 2017, *arXiv:1707.01083*.

[37] H. Xia and X. Gao, "Multi-scale mixed dense graph convolution network for skeleton-based action recognition," *IEEE Access*, vol. 9, pp. 36475–36484, 2021, doi: 10.1109/ACCESS.2020.3049029.

[38] J. Liu, A. Shahroudy, M. Perez, G. Wang, L.-Y. Duan, and A. C. Kot, "NTU RGB+D 120: A large-scale benchmark for 3D human activity understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 10, pp. 2684–2701, Oct. 2020, doi: 10.1109/TPAMI.2019.2916873.

[39] A. Paszke, "PyTorch: An imperative style, high-performance deep learning library," 2019, *arXiv:1912.01703*.

[40] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*. London, U.K.: Chapman & Hall, May 1994., doi: 10.1201/9780429246593.

[41] M. Li, S. Chen, X. Chen, Y. Zhang, Y. Wang, and Q. Tian, "Actional-structural graph convolutional networks for skeleton-based action recognition," 2019, *arXiv:1904.12659*.

[42] X. Gao, W. Hu, J. Tang, J. Liu, and Z. Guo, "Optimized skeleton-based action recognition via sparsified graph regression," 2019, *arXiv:1811.12013*.

[43] C. Si, W. Chen, W. Wang, L. Wang, and T. Tan, "An Attention enhanced graph convolutional LSTM network for skeleton-based action recognition," 2019, *arXiv:1902.09130*.

[44] K. Hu, Y. Ding, J. Jin, L. Weng, and M. Xia, "Skeleton motion recognition based on multi-scale deep spatio-temporal features," *Appl. Sci.*, vol. 12, no. 3, p. 1028, Jan. 2022, doi: 10.390/app12031028.

[45] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[46] M. Lindauer and F. Hutter, "Best practices for scientific research on neural architecture search," 2020, *arXiv:1909.02453*.

FELIX TEMPEL received the M.Sc. degree in medical engineering. He is currently pursuing the Ph.D. degree in computer science with the Norwegian University of Science and Technology.

ESPEN ALEXANDER F. IHLEN received the M.Sc. degree in human movement science and the Ph.D. degree in medical technology. He is currently a Senior Researcher of human movement science with the Norwegian University of Science and Technology. He is a Research Coordinator and a Data Scientist with Initial Force AS.

INGA STRÜMKE received the M.Sc. degree in theoretical physics and the Ph.D. degree in particle physics. She is currently an Associate Professor of artificial intelligence with the Norwegian University of Science and Technology.

• • •