AG-NAS: An Attention GRU-based Neural Architecture Search for Finger-Vein Recognition

Huafeng Qin, Chao Fan, Shaojiang Deng, Yantao Li, Mounim A. El-Yacoubi, and Gang Zhou, *Fellow, IEEE*

***Abstract*—Finger-vein recognition has attracted extensive attention due to its exceptional level of security and privacy. Recently, deep neural networks (DNNs), such as convolutional neural networks (CNNs) showing robust capacity for feature representation, have been proposed for vein recognition. The architectures of these DNNs, however, have primarily been manually designed based on human prior knowledge, which is both time-consuming and error-prone. To overcome these problems, we propose AG-NAS, an Attention Gated recurrent unit-based Neural Architecture Search to automatically search for the optimal network architecture, thereby improving the recognition performance for different finger-vein recognition tasks. First, we combine the self-attention mechanism and gated recurrent unit (GRU) to propose an attention GRU module employed as a controller to generate the architectural hyper-parameters of candidate neural networks automatically. Second, we investigate a parameter-sharing supernet policy to reduce the search space, computation, and time costs. Finally, we conduct rigorous experiments on our finger-vein database and two public finger-vein databases. The experimental results demonstrate that the proposed AG-NAS outperforms the representative approaches and achieves state-of-the-art recognition accuracy.**

***Index Terms*—Finger-vein recognition, Deep learning, Neural architecture search (NAS), Gated recurrent unit (GRU), Self-attention**

I. Introduction

As a result of the dramatic deployment of the Internet, IoT, and wearable devices, along with the ever-increasing personal data by these technologies, information security has become the focus of unprecedented research efforts in the past decade. Traditional identification methods, such as magnetic cards, keys, passwords, or personal identification numbers (PINs), are vulnerable to stealing, copying, and forging, as well as to various attack methods, such as smudge attacks [1] and shoulder-surfing attacks [2]. To address these shortcomings, biometrics has been widely investigated due to the following advantages: 1) Convenience: As an individual's inherent modality, biometric traits can be automatically verified in less than one second, and they are difficult to forget or cannot be lost. 2) Security: Biometric traits are resistant to duplication and forgery. Therefore, biometric traits, such as faces and fingerprints, have been investigated for identification/verification in recent years, as they are potential candidates to replace traditional identification methods. Biometric traits include physiological traits and behavioral traits. Physiological traits are more readily collectible and have been widely used in practical scenarios, such as door access and mobile payments, while behavioral traits require highly configured devices to capture motion patterns.

Physiological biometrics can be divided into two categories: 1) Extrinsic traits, such as face [3] and fingerprint [4], and 2) Intrinsic traits, such as palm vein [5], hand vein [6], and finger vein [7–9]. Extrinsic traits have been applied in different scenarios, such as immigration clearance, financial payments, access control systems, and consumer electronic products. However, they are vulnerable to attacks and may be copied without users' permission [10, 11], which raises concerns about security and privacy. Intrinsic traits, by contrast, are concealed in our bodies and have two advantages [6, 12]: 1) Liveness detection: Vein patterns can only be collected from living individuals; 2) High security and privacy: Blood vessels are naturally concealed beneath the human skin from birth, remaining invisible to the naked eye by visible light. It is hard to replicate vein patterns without users' awareness. Moreover, it is difficult to forge vein patterns to attack vein recognition systems. Finger-vein biometrics has received, as a result, increasing attention in recent years. Like facial and fingerprint recognition systems, most vein recognition systems share the same mechanism: physiological data collection, feature extraction, and feature matching for recognition.

Finger-vein recognition remains a challenging task as image acquisition is affected by various factors, such as lighting [13], temperature [14], light scattering [15], and user behavior [14]. These factors cause noise and irregular shadow regions in the captured images, ultimately degrading recognition accuracy. To solve this problem, researchers have proposed various algorithms based on traditional machine learning and other technologies to improve the robustness. Deep learning technologies, including convolutional neural networks (CNNs), have recently demonstrated remarkable performance across diverse tasks, such as image classification and object detection [16–20]. Inspired by this success, deep learning has been

Huafeng Qin is with the School of Computer Science and Information Engineering, Chongqing Technology and Business University, Chongqing 400067, China (e-mail: qinhuafengfeng@163.com).

Chao Fan, Shaojiang Deng, and Yantao Li are with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: {202014021017,sj_deng,yantaoli}@cqu.edu.cn).

Mounim A. El-Yacoubi is with SAMOVAR, Telecom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France (e-mail: mounim.el_yacoubi@telecom-sudparis.eu).

Gang Zhou is with the Department of Computer Science, William & Mary, Williamsburg, VA 23185, USA (e-mail: gzhou@cs.wm.edu).

Manuscript received June XX, 2023; revised XXXX XX, 201X. This work was supported in part by the National Natural Science Foundation of China under (Grant Nos. 61976030, 62072061, and U20A20176), the Scientific Innovation 2030 Major Project for New Generation of AI (Grant No. 2020AAA0107300), the Fellowship of China Post-Doctoral Science Foundation (Grant No. 59676651E), the Science Fund for Creative Research Groups of Chongqing Universities (Grant Nos. CXQT21034, KJQN201900848, and KJQN201500814), and the National Key R&D Program of China (Grant No. 2022YFC3801703). (Corresponding author: Yantao Li.)

introduced to the field of vein recognition [5, 5, 7, 8, 21–24, 24–31], as indicated by the growing trend in recent years. Current deep neural network (DNN) architectures for vein recognition, nonetheless, have mainly been devised manually by human experts, which raises the following issues: 1) Limited exploration of the design space: Human experts may not be able to explore the entire design space of DNN architectures due to their limited knowledge and experience, especially for complex neural architectures. In addition, they may have biases towards specific architectures they have worked or were familiar with, which may limit the diversity of the architecture exploration; 2) High time consumption and cost: Manually developing CNN architectures is a time-consuming and costly process, requiring significant expertise and resources; 3) Difficulty in optimizing hyperparameters: Optimizing hyperparameters for CNN architectures can be a challenging task, and human experts may not always be able to find the optimal hyperparameters; 4) Lack of scalability: Manually designed architectures may not be scalable to other datasets or more complex tasks, limiting their usefulness in real-world applications. To overcome these shortcomings, neural architecture search (NAS) technology [32–34], aiming to automatically find the optimal neural network architecture for a given task, has received more and more attention. Compared to traditional approaches for designing neural networks, NAS automates this process using machine learning algorithms to explore the space of possible architectures and find the best-performing ones. NAS demonstrates clear potential in enhancing the performance of DNNs and streamlining the design process, thereby reducing the time and effort involved [35]. NAS has been employed in various fields, including image classification [36], natural language processing [37], and speech recognition [38].

To enhance recognition performance, we propose AG-NAS, an Attention Gated recurrent unit-based Neural Architecture Search to automatically generate the optimal network architecture for a given finger-vein classification task, as an extension of our preliminary work [39]. The proposed AG-NAS comprises two crucial modules: a controller and a supernet. The controller is responsible for generating the architectural hyperparameters. The supernet houses large candidate networks that share parameters to reduce the search space, computation, and time costs, thereby improving search efficiency.

The main contributions of our work can be summarized as follows:

1) We propose AG-NAS, an attention gated recurrent unit-based neural architecture search for finger-vein recognition. Specifically, we design an attention gated recurrent unit by introducing the attention mechanism into the recurrent network, thereby making it capable of learning long-range dependencies within sequences and enabling AG-NAS to find an optimal network architecture automatically for finger-vein recognition.

2) We design a parameter-sharing supernet to improve the search efficiency. Specifically, the controller outputs the architectural descriptor, based on which the candidate subnets are determined from the supernet. Since the parameters are shared among these subnets, AG-NAS can achieve robust empirical performance using a few GPU hours.

3) We conduct rigorous experiments on our finger-vein database and two public finger-vein databases to evaluate the search efficiency and the performance of AG-NAS. The experimental results demonstrate that the deep neural network generated by AG-NAS outperforms the representative (deep learning-based and NAS-based) finger-vein recognition approaches in terms of EER and accuracy.

The remainder of this paper is organized as follows. In Section II, we introduce the related work on finger-vein classification. In Section III, we detail the proposed AG-NAS and evaluate the performance of AG-NAS in Section IV. Finally, Section V concludes this work.

II. Related Work

Finger-vein recognition algorithms are designed to verify or identify individuals by analyzing their captured finger-vein patterns. Due to the unique and stable vein texture information in finger-vein images, finger-vein recognition offers high accuracy, security, and robustness. As a result, it has found widespread applications in real-world scenarios, including financial payments, border inspections, and access control systems. To enhance the accuracy of finger-vein recognition, various methods have been introduced to extract robust features. These can be categorized into traditional vein recognition methods, deep learning-based vein recognition methods, and neural architecture search.

A. Traditional Vein Recognition Methods

Traditional vein recognition approaches rely on handcrafted and shallow learning methods to extract vein patterns. The handcrafted methods use manually designed descriptors to extract vein patterns, including curvature-based approaches [40–45], Gabor filter-based approaches [14, 15, 46], and local binary descriptor-based approaches [47, 48]. The shallow learning methods exploit traditional machine learning techniques to model the input. Representative approaches, such as k-means clustering [49], SVM [50], PCA [51], 2D-PCA [52], and Sparse Representation (SR) [53], have been proposed to learn vein feature representation. To improve performance, LRR (Low-Rank Representation) [54] is used for discriminative feature extraction by adding a regularization term to constrain low-rank coefficients, thereby enhancing the discriminative power.

B. Deep Learning-based Vein Recognition Methods

Deep learning-based methods refer to machine learning technologies employing neural networks with many layers, facilitating a hierarchical, non-linear mapping of input data that directly operates on raw data. Recent deep learning models, such as CNNs and Transformers, have shown robust feature representation capability and have proven their effectiveness in diverse computer vision tasks [18–20]. Inspired by this success, some researchers have applied deep learning in vein image quality assessment [55–57], vein texture segmentation [7, 21–23], and vein recognition [5, 5, 8, 24, 24–31]. For

example, FV-CNN [24] is a specially designed CNN for finger-vein recognition. PV-CNN [5] uses MSMD-GAN to generate augmented data for single-sample palm-vein recognition. To improve computation efficacy, LightweightDeepCNN [28] proposes a lightweight vein recognition algorithm relying on a triplet loss function to train the model. To extract robust features, Arcvein [30] proposes a loss function called the cosine center loss to learn both inter-class and intra-class information to improve the discriminating ability of CNNs for finger-vein verification. FVRASNet [29] proposes a lightweight CNN model that integrates recognition and anti-spoofing tasks into a unified CNN model through multi-task learning methods. Recently, transformer-based approaches have been investigated to learn long-range dependency features for vein recognition. Typical works [25, 26, 31] have shown promising performance for vein recognition.

C. Neural Architecture Search

Recently, neural architecture search [32] has been proposed to automatically search for an optimal neural network architecture using reinforcement learning techniques. NAS has found widespread applications in various fields, such as computer vision [58], natural language processing [59], and speech recognition [60]. As the neural network architecture is determined by automatic search policy instead of prior knowledge, it can significantly improve classification performance and efficiency. For example, reinforcement learning is employed to automatically find CNN architectures for a given learning task in [61]. To reduce the computation cost on large datasets, Zoph et al. [62] propose to search for an architectural building block on a small dataset and then transfer the block to a larger dataset, designing a new search space to enable transferability. For the same purpose, Pham et al. [58] propose an efficient neural architecture search (ENAS) for automatic model design, and Cai et al. [34] introduce ProxylessNAS to learn the architectures for large-scale target tasks and target hardware platforms directly. Liu et al. [33] propose the DARTS method to search for the architecture in a differentiable manner instead of searching it in a discrete set of candidate architectures, which is required to choose only one path during the verification phase, inevitably leading to a gap between neural architectures during the search and verification phases. To address this issue, Chang et al. [63] investigate the differentiable architecture approximation (DATA) based on an ensemble Gumbel-softmax (EGS) estimator to automatically approximate architectures during searching and validating in a differentiable manner. Jiang et al. [64] improve the differentiable architecture search by removing the softmax-local constraint for named entity recognition (NER). In addition, Li et al. [65] modularize the large search space of NAS into blocks to ensure that the potential candidate architectures are effectively trained, which reduces the representation shift caused by the shared parameters and leads to the correct rating of the candidates. To overcome the catastrophic forgetting in one-shot NAS, Zhang et al. [66] formulate the supernet training as a constrained optimization problem of continual learning and propose a search-based architecture selection (NSAS) loss function for a greedy search of the most representative subset. Lu et al. [67] propose a differentiable NAS to decrease the uncertainty of differentiable architecture search. To reduce time cost, Mellor et al. [68] examine the overlap of activation between data points in an untrained network and compute a score based on a predefined kernel matrix, which is subsequently incorporated into a simple algorithm to search for robust networks, all without the need for training. Inspired by recent NAS approaches, Kim et al. [69] introduce a novel NAS approach to search for an optimal architecture for spiking neural networks (SNNs). Jia et al. [36] survey the NAS technology and explore NAS-based 2D and 3D palmprint and palm-vein recognition, where they assess the recognition performance of twenty representative NAS methods on palmprint and vein databases in their experiments.

Handcrafted techniques are rule-based or expert-based approaches designed based on prior knowledge and heuristics for specific tasks. Their performance, therefore, is very limited. Shallow learning approaches, such as SVM and PCA, can be regarded as shallow neural networks with one or two layers, and they usually fail, as a result, to extract high-level features. Deep learning approaches, by contrast, stack multiple layers to form a DNN, which allows the extraction of rich non-linear hierarchical representations when trained on large datasets. Manually designed deep networks, however, are time-consuming, expensive, and not generalizable to different tasks. NAS is an effective solution that has been widely investigated in recent years and recently brought into the vein recognition task. Some of the proposed NAS-based approaches, nonetheless, employ traditional approaches, such as the recurrent neural network, as a controller for architecture searching. Hence, the controller's representation capacity is limited, degrading the search performance and inducing high time costs.

III. Method

In this section, we first introduce our AG-NAS. Then, we describe how to generate a deep neural network architecture by AG-NAS. Next, we detail the subnet generation, controller design, supernet design, and reinforcement learning, respectively. Finally, we discuss the training and testing of AG-NAS to ensure a comprehensive understanding of the algorithm.

A. AG-NAS

Deep neural networks (DNNs) are capable of automatically learning effective features from a large number of data. However, the application of DNNs to finger-vein recognition encounters several challenges, particularly in designing an appropriate network architecture for a given vein recognition task. In this work, we propose an attention GRU-based neural architecture search to automatically explore and optimize the neural network architectural hyperparameters in the context of vein recognition.

We illustrate the framework of the proposed AG-NAS in Fig. 1. As illustrated, the supernet consists of N block layers, with each providing M candidate operators (nodes). AG-NAS utilizes a controller to generate architecture descriptors, which

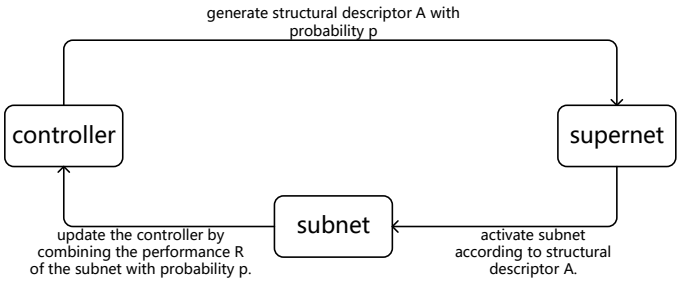


Fig. 1: Framework of AG-NAS.

activate specific nodes in the supernet to form subnets. These subnets undergo training on the training set, and the accuracy on the validation set serves as the reward signal for optimizing the controller's parameters. Throughout the training process, the controller and subnets are iteratively trained, leading to the determination of the optimal network architecture. Specifically, for the subnet training, the controller takes a set of random vectors as the input, and produces the probability p_s. A set of architecture descriptors A_s are then sampled based on the probability p_s. The supernet utilizes these architecture descriptors A_s to select one of the subnets as the active network, which is subsequently trained on the finger-vein training set to update its parameters. This process allows us to train S subnets for S input vectors. For the controller training, its output is regarded as the probability p_e, and a set of architecture descriptors A_e are sampled with the probability p_e. The active network is determined from the supernet using the architectural descriptor A_e. The accuracy of the active network on the validation set serves as the reward R, and this reward is used to compute the gradient of p_e optimizing the controller's parameters. To achieve stable training, E networks are generated based on E architectural descriptors from the controller, and their average accuracy is employed as a reward for controller training. After the iterative training, the controller outputs the architectural descriptor A' with the highest probability for each layer. An optimal subnet is then extracted from the supernet using the resulting architectural descriptor A' for finger-vein recognition.

B. Subnet Generation

In our approach, the selection of an activated subnet from a supernet is treated as choosing a path from the root node to a leaf node on a complete N-ary tree. Each node on this path corresponds to a specific architecture of one layer. The AG-NAS controller produces an architectural descriptor that outlines the selection process for its child nodes, designating the active node on the path. To activate nodes for subnet generation, each architectural descriptor must make three key decisions:

1) Determine whether to select the direct path, utilizing the output of the previous layer instead of the current layer as the input for the next layer;
2) If not choosing the direct path, decide the type and parameters of the network architecture in the current layer;
3) If not choosing the direct path, decide whether to use a shortcut path.

The three decisions are crucial for the performance of subnets. First, the direct path allows AG-NAS to automatically determine whether to reduce the number of layers. Second, the selection of the network architecture type and hyperparameters can significantly influence subnet performance. In our work, these encompass the type, number, and size of convolution kernels. Finally, like residual connection, the inclusion of a shortcut path enhances the information propagation within the subnet, alleviating issues of information loss and vanishing gradient to improve overall performance. Therefore, during the generation of architecture descriptors, the AG-NAS controller must carefully consider the interaction and balance among these three decisions. In our work, the architecture descriptor is defined as a triplet $[direct, param, shortcut]$, where $direct$ indicates the direct connection, $param$ represents the network hyperparameters, and $shortcut$ denotes the shortcut connection.

Fig. 2 depicts an example of subnet generation from the supernet using the controller. In this DNN, there are N hidden layers, each containing two alternative convolutional blocks: a conventional convolutional block and a separable convolutional block [70]. The N random vectors $x_1, x_2, ..., x_N$ are individually fed to the controller, producing corresponding outputs $h_1, h_2, ..., h_N$ that form an architecture descriptor. This descriptor, represented by the red connection lines in Fig. 2, determines the subnet. The architecture selection of each hidden layer is illustrated as follows:

1) At time t_1, the controller receives a seed vector x_1 and initial state vector h_0 as input, and generates the state vector h_1. This vector is then transformed into a probability triplet $a_1 = [a_{11}, a_{12}, a_{13}]$ through a non-linear fully connected layer. Using this probability triplet, the controller produces a triplet $d_1 = [direct_1, param_1, shortcut_1]$ for the first layer by randomly sampling based on the resulting probability triplet. Specifically, $direct_1 = 0$ or 1 indicate that the direct path is not selected or selected. $param_1 = c$ $(c = 1, ..., C)$ or $param_1 = [0, 0, ..., 1, ..., 0, 0] \in R^{1\times C}$ (where only the cth value in the one-hot vector $param_1$ is not equal to 1) indicates the selection of the cth type of network architecture, where C is the number of network architecture types ($C = 6$ in our experiments). $shortcut_1 = 1$ or 0 indicate that a shortcut path is used or not used. Therefore, the triplet $[0, 2, 1]$ implies that the second operator type and shortcut path for this layer are activated in the first layer as the data propagation passageway.
2) At time t_N, the controller takes the seed vector x_N and state vector h_{N-1} from the previous layer as inputs, and generates the state vector h_N. Similarly, using the probability triplet $a_N = [a_{N_1}, a_{N_2}, a_{N_3}]$, the controller outputs the triplet $d_N = [direct_N, param_N,$

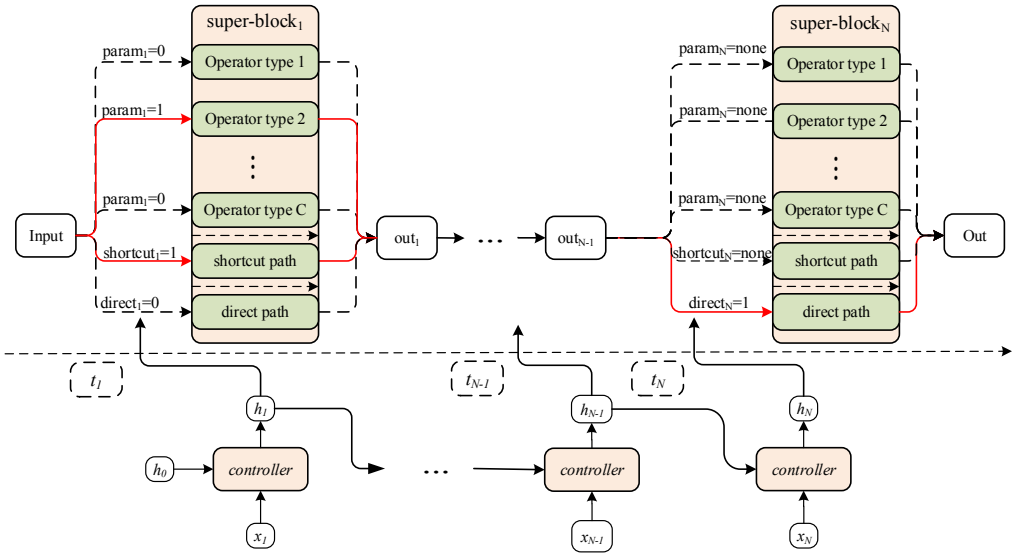


Fig. 2: Example of subnet generation from the supernet. The red line connection nodes denote the activated ones, emphasizing the generated subnet. The controller takes a sequence $x_1, x_2, ..., x_N$ and h_0 as its input, and forwards the corresponding output sequence $h_1, h_2, ..., h_N$ to the supernet. The network architecture type and hyperparameters for each layer are then determined for the purpose of classification.

$shortcut_N]$ for the Nth layer. As shown in Fig. 2, in the final layer, $direct_N = 1$ indicates the selection of the direct path. As a result, the output of the $(N-1)$th layer is directly used as the output of the Nth layer, and two parameters $param_N$ and $shortcut_N$ are not used or $none$. Thus, in the Nth layer, the direct path is activated as the pathway for data propagation.

Following N iterations, the controller produces N triplet sets $d_1, d_2, ..., d_N$, forming an architectural descriptor A. This descriptor is employed to activate nodes within the supernet, yielding a subnet (as depicted by the red connection lines in Fig. 2). The search method, orchestrated by the collaboration between the controller and supernet, exhibits the flexibility to generate diverse neural network architectures, automatically optimized for distinct vein recognition tasks.

C. Controller

In NAS, the controller plays a crucial role in generating candidate network architectures and updating parameters based on the feedback reward signals. Typically, a recurrent neural network (RNN), such as LSTM, is employed for sequence data representation learning. Due to its fewer parameters and faster computation speed, GRU has gained widespread application in various practical contexts. GRU encompasses two critical gates: the update gate and the reset gate. The update gate controls how much information from the previous hidden state is retained in the current time step, while the reset gate determines how much information from the previous hidden state is used to compute the candidate's hidden state at the current time step. The GRU gate mechanism effectively addresses issues of gradient vanishing and exploding, enabling the network to adeptly process long sequence data. Recently, transformers have demonstrated significant success in natural language processing and computer vision. Thanks to their self-attention mechanism, transformers excel in capturing long-range dependencies between different positions in an input sequence. Inspired by its robust feature representation, we integrate the attention mechanism into GRU, proposing an attention GRU as the controller for our neural network architecture search.

Architecture: In our approach, the controller comprises two stacked attention GRUs and a classifier group, capable of transforming the hidden state into a probability triplet. The resulting triplet contains three elements: $direct$, $param$, and $shortcut$, representing the operators of the direct path, network architecture type and parameters, and shortcut path, respectively. Due to the distinct dimensions of these three elements, the classifier group employs varied processing operators. For $direct$ and $shortcut$, the fully connected layers and sigmoid activation function are used for feature extraction, and the classifier group uses the fully connected layers and softmax layer for $param$.

Attention GRU: As shown in Fig. 3, the attention GRU is introduced by substituting the operations in GRU with self-attention and cross-attention to achieve update and reset gates. Specifically, considering x_t as the input vector and h_{t-1} as the previous hidden state, we calculate the query matrix Q_x, key matrix K_x, and value matrix V_x for x_t. Similarly, we obtain the query matrix Q_h, key matrix K_h, and value matrix V_h for h_{t-1}. The self-attention of input vector x_t can be computed by Eq. (1):

$$\begin{aligned} SA_x &= Attention(Q_x, K_x, V_x) \\ &= Softmax(\frac{Q_x K_x^T}{\sqrt{d_x}} + B_x)V_x. \end{aligned} \tag{1}$$

Similarly, we compute the self-attention of the hidden state

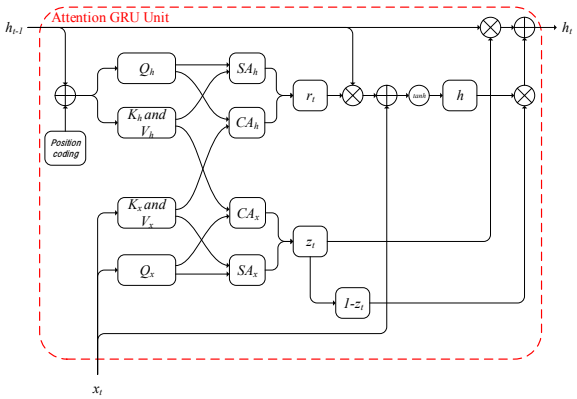


Fig. 3: Architecture of the attention GRU.

h_{t-1} and the cross-attention between x_t and h_{t-1} by Eqs. (2)-(4):

$$CA_x = Attention(Q_x, K_h, V_h), \tag{2}$$

$$SA_h = Attention(Q_h, K_h, V_h), \tag{3}$$

$$CA_h = Attention(Q_h, K_x, V_x), \tag{4}$$

where CA_x and CA_h represent the cross-attention between x_t and h_{t-1}, and SA_h indicates the self-attention of the hidden state x_t. The update gate z_t is determined based on SA_x and CA_x in Eq. (5):

$$z_t = \sigma(Linear(Concat(SA_x, CA_x))). \tag{5}$$

Similarly, the reset gate r_t is computed based on SA_h and CA_h in Eq. (6):

$$r_t = \sigma(Linear(Concat(SA_h, CA_h))), \tag{6}$$

where σ represents the sigmoid function, $Linear$ indicates a linear mapping function, and $Concat$ denotes a vector concatenation function. Using z_t and r_t, we compute the candidate hidden state h by Eq. (7):

$$h = tanh(Linear(x) + Linear(r_t \odot h_{t-1})), \tag{7}$$

where $\odot$ denotes the element-wise multiplication operation. The final hidden sate h_t is calculated by Eq. (8):

$$h_t = (1 - z) \odot h_{t-1} + z \odot h. \tag{8}$$

In AG-NAS, the controller takes a seed vector x_t and the previous hidden state h_{t-1} as inputs at time t, generating a new hidden state h_t. Then, the classifier group transforms h_t into a probability triplet for the network architecture of the tth layer. This process is iteratively repeated by taking h_t as the input to AG-NAS at time step $(t + 1)$.

D. Supernet

Traditional NAS methods typically involve a controller generating architecture descriptors, followed by the creation and training of a new DNN. The accuracy of the validation set is then utilized as a reward to train the controller. These approaches, however, come with high time consumption and require substantial hardware resources for each search. As a result, the application of traditional NAS methods to large-scale network search tasks becomes challenging. To solve this issue, we design a supernet for optimal subnet search, wherein the controller generates the descriptors to determine the subnet's architecture. The nodes of the supernet are activated layer by layer based on these descriptors, resulting in the generation of a subnet. This approach employed by AG-NAS eliminates the need to create and train a new subnet for each search, thereby reducing time costs.

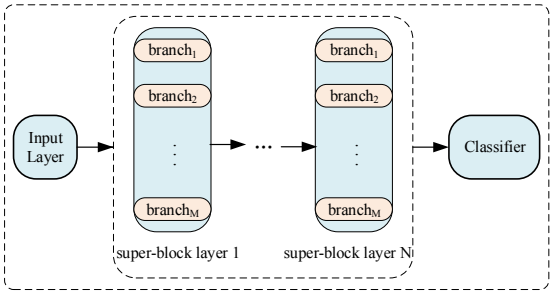


Fig. 4: Architecture of the supernet.

TABLE I: Optimal architecture of super-block layer

Architecture	Kernel size
conv-relu-norm	(3, 3)
conv-relu-norm	(5, 5)
separable conv-relu-norm	(3, 3)
separable conv-relu-norm	(5, 5)
average pool	(2, 2)
max pool	(2, 2)
shortcut	-
direct path	-

We illustrate the supernet architecture in Fig. 4. As shown, the supernet architecture comprises three modules. The first module consists of an input layer with a 3×3 convolutional layer and a batch normalization layer, mapping the input sequence to a 32-channel high-dimensional space. The second module comprises N $(N = 10)$ super-block hidden layers, each containing M nodes or selectable network architectures, where $M = C + 2$ and C is the number of operator types (as shown in Fig. 2). During the training process, AG-NAS activates only one of the nodes as the active node for the current layer. In our experiments, M is set to 8 (2 convs, 2 separable conv, 1 average, 1 max pooling, 1 shortcut, and 1 direct patch), indicating that there are 8 selectable network architectures for each hidden layer (as detailed in Table I). Note that we select only one of the first 6 operator types (2 convs, 2 separable conv, 1 average, and 1 max pooling) for each hidden layer, and the remaining nodes are not activated when the direct patch is activated. This way, for each layer, there are only 13 possible combinations of the operator type, shortcut, and direct patch. Hence, approximately 13^{10} subnets are included in our search space. Finally, the third module consists of a classification layer that outputs the probability of the input belonging to all classes. As the first module of the entire network, the input layer aims to extract coarse features and reduce the input dimension. The super-block layers constitute the key modules

of the supernet. Each node in the super-block layer represents a candidate network architecture, and the architecture of the network in each layer is determined by activating these nodes. The classifier layer takes the features extracted by the super-block layer as input and predicts its probability of belonging to all classes, achieving the final classification task. Overall, AG-NAS employs the designed supernet to automatically determine an optimal network architecture for a specific task through node activation. This approach provides flexibility and adaptability for different finger-vein recognition tasks.

E. Reinforcement Learning

In our work, we utilize the cross-entropy function to calculate the loss and optimize the parameters of the supernet ω for generating the subnet weights. Specifically, the supernet activates the subnet to execute the finger-vein recognition task based on a given architectural descriptor. Subsequently, we optimize the parameters of the subnet ω by computing the loss gradient through forward and backward propagation on the target task. The controller produces a sequence representing a candidate network architecture $(a_1, a_2, ..., a_N)$. This candidate architecture undergoes training on the training set, and the resulting accuracy on the validation set serves as a reward signal R. This reward is used to update the controller's parameters, which in turn generate the architectural hyperparameters of neural networks. Concretely, in the search for an optimal architecture, the controller maximizes the expected reward $J(\theta_t)$ as computed by Eq. (9):

$$J(\theta_t) = E_{P(a_1:a_N;\theta_t)}[R] = \sum_{a_{1:N} \in S} P_{\theta_t}(a_{1:N}) \cdot R. \quad (9)$$

However, there is no function relationship between reward R and controller parameters θ, implying that the gradient cannot be propagated through the chain rule. To address this issue, the REINFORCE algorithm [71], as a policy gradient algorithm, can be used to maximize the expected reward of the controller. Specifically, the formula for updating the controller using the REINFORCE algorithm is given by Eq. (10):

$$\begin{aligned} \nabla J(\theta_t) &= \sum_{a_{1:N} \in S} \nabla P_{\theta_l}(a_{1:N}) \cdot R \\ &= \sum_{a_{1:N} \in S} P_{\theta_t}(a_{1:N}) \frac{\nabla P_{\theta_t}(a_{1:N})}{P_{\theta_t}(a_{1:N})} \cdot R \\ &= \sum_{a_{1:N} \in S} P_{\theta_t}(a_{1:N}) \nabla_{\theta_t} log P_{\theta_t}(a_{1:N}) \cdot R \\ &= E[\nabla_{\theta_t} log P_{\theta_t}(a_{1:N}) \cdot R]. \end{aligned} \quad (10)$$

It is an unbiased estimate for our gradient, and its empirical approximation is expressed by Eq. (11):

$$\mathbb{L}_{\theta_t} \approx -log P_{\theta_t}(a_{1:N}) \cdot R = -\sum_{n=1}^{N} log P_{\theta_t}(a_n) \cdot R. \quad (11)$$

The optimizing function given in Eq. (9) is equivalent to optimizing the following problem, as shown in Eq. (12):

$${\theta_l}^* = argmin_E \mathbb{L}_{\theta_t} = argmin_{\theta_t} log P_{\theta_t}(a_{1:N}) \cdot R. \quad (12)$$

This problem is solved using the Adam optimizer with a learning rate α and batch size B. The training procedure for each batch is given by Eq. (13):

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{M} \sum_{m=1}^{B} \nabla_{\theta_t} \mathbb{L}_{\theta_t}. \quad (13)$$

By combining Eq. (12) and Eq. (13), we can obtain Eq. (14):

$$\theta_{t+1} = \theta_t + \alpha \frac{1}{M} \sum_{m=1}^{B} \sum_{n=1}^{N} \nabla_{\theta_t} log P_{\theta_t}(a_n) \cdot R_m. \quad (14)$$

To reduce the variance during the training process, the parameter b is introduced into Eq. (14) to yield Eq. (15):

$$\theta_{t+1} = \theta_t + \alpha \frac{1}{B} \sum_{m=1}^{B} \sum_{n=1}^{N} \nabla_{\theta_t} log P_{\theta_t}(a_n) \cdot (R_m - b), \quad (15)$$

where N denotes the number of layers in the generated network architecture, and b is the moving average of the reward r_m over a certain period of time. The term $P_{\theta_t}(a_n)$ represents the conditional probability of the controller selecting network architecture a_n at the nth layer, given the previous architectures $a_{(n-1):1}$. The parameter B controls the training frequency of the controller, indicating that the gradient is accumulated over a span of B runs, and the controller's parameters are updated by averaging the accumulated gradients.

Based on reinforcement learning, we propose a dynamic $\varepsilon - greedy$ strategy for controller training. The $\varepsilon - greedy$ strategy is a widely adopted and effective approach in reinforcement learning. It randomly generates an action descriptor with probability ε while selecting an action from its probability vector with probability $1 - \varepsilon$. Typically, the value of ε is set to a small value, such as 0.1 or 0.2, ensuring that the controller mostly chooses actions based on its learned policy but still has a chance to explore other actions. This balance between exploration and exploitation of historical experience allows the controller to take new actions while continuously utilizing its learned knowledge. When ε equals 0, the controller selects actions based solely on its learned policy. Conversely, when ε equals 1, the controller randomly selects actions without considering its learned policy. Existing works [32, 72] often use a fixed value of ε for training, which may not be effective at different stages of the learning process, resulting in either excessive exploration or exploitation. To address this issue, we propose a dynamic $\varepsilon - greedy$ strategy to train the controller, where $\varepsilon = 1 - Maccuracy$ and $Maccuracy$ is the moving average of the accuracy for the generated network at different times. Specifically, if the performance of the generated network is poor, the controller's policy is unstable and requires more exploration, thus increasing ε. In contrast, if the performance is good, indicating a relatively stable controller's policy that can exploit more knowledge, ε decreases. The dynamic $\varepsilon - greedy$ strategy automatically adapts ε to balance exploration and exploitation at different stages during the learning process.

F. Model Training and Testing

AG-NAS consists of two main blocks: the controller and the supernet, which are iteratively trained to search for an optimal network architecture. The controller's parameters are denoted as θ, and the supernet's parameters are denoted ω, with shared weights among subnets. The training process of AG-NAS can be divided into three stages: warm-up stage, main training stage, and generation stage. In the warm-up stage, the supernet is trained to provide improved initial values for the optimal network search. In the main training stage, the optimal subnet is searched through iterative training of the controller and the subnet. In the generation stage, an optimal neural network is obtained for vein classification. The detailed training process is as follows:

1) Warm-up stage: We keep the parameters of the controller θ fixed and proceed to train the supernet ω. Throughout this training process, the controller with parameters θ generates a set of random network architectural descriptors. These descriptors are used to determine various subnets from the supernet, and the resulting subnets are subsequently trained using the training dataset.
2) Main training stage: Both the parameters of the controller θ and the supernet ω are iteratively updated.
   - To update the parameters of the supernet ω, we maintain the parameters of the controller θ fixed. In each training step, the controller generates a set of architecture descriptors, from which a subnet is determined within the supernet. The parameters of the supernet ω are then updated by training the subnet on the training dataset. Note that if the parameters of the nodes have been trained in the warm-up stage, they will be fine-tuned during this stage.
   - To update the parameters of the controller θ, we keep the parameters of the supernet ω fixed. Throughout the training process, the controller generates a set of architecture descriptors to construct a subnet from the supernet. The accuracy of the resulting subnet on the validation set is employed as the reward R. After repeating $E(E = 10)$ steps, E subnets are obtained. The average accuracy of E subnets is computed to update the parameters of the controller θ.
3) Generation stage: Following the training, the controller produces the architecture descriptor, from which an optimal subnet is generated within the supernet. This subnet is utilized as the classifier and further trained on the training set for finger-vein recognition. Finally, the trained classifier is employed for testing.

The optimization process of the proposed AG-NAS is summarized in Algorithm 1.

Algorithm 1 Optimizing the proposed AG-NAS.

Input: Initial parameter of controller θ; Parameter of supernet ω; Training set X_t and label L_t; Validation set X_v and label L_v;
Output: Target network;

For $1 \leq$ e $\leq warm\_epoch$;
  Generate architecture descriptors by the controller;
  Active a subnet from the supernet using architecture descriptors;
  Train the resulting subnet using the training dataset and update the parameters of supernet ω;
end
For $1 \leq t \leq epoch$;
  For $1 \leq t_1 \leq 10$;
  Generate architecture descriptors by the controller;
  Active a subnet of the supernet using architecture descriptors;
  Train the resulting subnet using the training dataset and update the parameters of supernet ω;
  end
  For $1 \leq t_1 \leq 500$;
  Generate architecture descriptors by the controller;
  Active a subnet from the supernet using architecture descriptors;
  Compute the accuracy of a subnet on validation set as the reward R;
    if t_1 Mod $E = 0$
    Update parameters of the controller according to Eq. (15) based on the average accuracy of E subnets;
    end
  end
end
Select an optimal subnet from the supernet as the target network based on the architecture descriptor.

IV. Performance evaluation

To evaluate the performance of AG-NAS, we conducted rigorous experiments on our database and two public finger-vein databases, using the PyTorch deep learning framework. The experiments were executed on a high-performance computer equipped with an Intel-10900X 3.7 GHz processor featuring 10 cores and 20 threads, 128GB memory, and an NVIDIA 3090 graphics card. In the experiments, AG-NAS is compared with classical neural network models, namely ResNet [19], VGGNet [20], and GoogLeNet [18], as well as the state-of-the-art vein classifiers, namely FV-CNN [24], PV-CNN [5], LightweightDeepCNN [28], Arcvein [30], and FVRASNet [29]. Note that we fine-tune the publicly provided pre-trained ResNet, VGG, and GoogLeNet models. In addition, NAS-based models, including ENAS [58], ProxylessNAS [34], and DU-DARTS [33], are considered in our comparative experiments.

A. Database

1) Database A: Currently, most existing works are tested based on finger-vein images collected through device prototypes rather than commercial sensors, making it challenging to fully evaluate the effectiveness of finger-vein recognition methods in practical applications. Moreover, the absence of public finger-vein databases collected by commercial sensors limits the development of real-world recognition systems. To

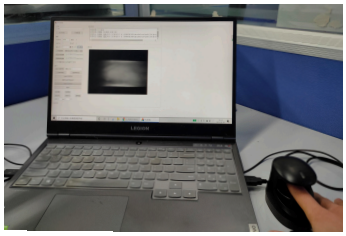
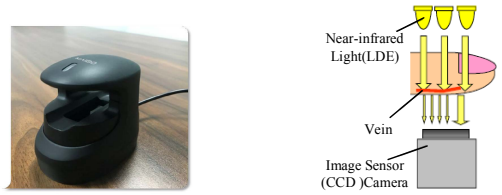


Fig. 5: Image capturing system



(a) (b)

Fig. 6: Vein image capturing device. (a) Finger-vein image capturing device; (b) Principle of image capturing

address this issue, we collaborated with Corespirit company [73] to develop a low-cost finger-vein capturing system, including both hardware and software components, as shown in Fig. 5. The hardware device comprises a light controlling unit (LCU) and an image collection unit (ICU). The LCU, equipped with a micro-controller unit (Advanced RISC Machine (ARM)), controls the light intensity of near-infrared LEDs, while the ICU consists of a near-infrared LED group, CCD camera, and NIR filter. The software comprises a light-controlling system, finger detection system, and recognition system. The light-controlling system adjusts brightness to capture high-quality vein images, the finger detection system identifies recognition request, and the recognition system handles image preprocessing, feature extraction, and identification. Our commercial finger-vein imaging sensor (Fig. 6(a)) features a USB for communication with the host computer, enabling data query, deletion, modification, registration operators, and access to the recognition results. The scanner connects to a PC or laptop via a USB interface for image access, with installed software (Fig. 5). During image collection, a finger is positioned between the NIR camera (lower side) and NIR illumination unit (upper side); the infrared light from the NIR illumination unit passes through the finger, and the camera records vein patterns, as shown in Fig. 6(b). In our experiments, we have constructed a finger-vein database [74] using commercial sensors, comprising 6,000 images (10 images $\times$ 6 fingers $\times$ 100 subjects) from 100 subjects (66 males and 34 females). Each subject provided six fingers, namely the index, middle, and ring fingers of both hands, each contributing 10 images. As the background holds no classification information, we cropped the images to extract the region of interest (ROI), and the resulting ROI images were further normalized to 55×127.

2) Database B: The Hong Kong Polytechnic University finger-vein image database (HKPU dataset) [14] is gathered from 156 participants using a non-contact imaging device.

(a) (b)

Fig. 7: Sample results from database A. (a) Original image; (b) Normalized gray image from (a).

(a) (b)

Fig. 8: Sample results from database B. (a) Original image; (b) Normalized gray image image from (a)

The first 105 participants contributed 2,520 images ((105×2) fingers $\times$ 6 images $\times$ 2 sessions), captured at two different times. For each participant, six images were provided for each of their index and middle fingers during each session, resulting in 24 images (6 images $\times$ 2 fingers $\times$ 2 sessions) for two sessions. The remaining 51 participants contributed only 612 images, exclusively collected during the first session. In our experiments, only the first 2,520 images captured at two sessions are used for performance evaluation. To facilitate matching, we extract the ROI region using the preprocessing approach [14] and normalize it to 49×181.

3) Database C: The MMCBNU6000 finger vein database [75] comprises 6,000 images (100 volunteers $\times$ 6 fingers $\times$ 10 images) from 100 volunteers. Each volunteer contributed 60 images (6 fingers $\times$ 10 images) captured from the index, middle, and ring fingers of both hands. Due to potential variations such as translation, rotation, scale, scattering, and uneven lighting in the collected images, which could degrade the verification performance, we extracted ROIs. The resulting ROI images were resized to 59×127. As a result, 6,000 ROI images were obtained.

Both the original images and the resulting ROI images for the three databases are displayed in Figs. 7, 8, and 9, respectively.

B. Experimental Settings

Each dataset is partitioned into the training set, validation set, and testing set. The training set is used to train the super network, while the validation set is utilized for assessing the subnets' performance within the super network and optimize the controller's parameters. The testing set is reserved for evaluating the performance of the optimal neural network model. For database A, which consists of 600 fingers from 100 subjects, treating each finger as a distinct class results in 600 classes. For each class, images are selected in a $4:2:4$ ratio for training, validation, and testing, respectively. Therefore, the training set comprises 2,400 images (4 images $\times$ 600 fingers), the validation set includes 1,200 images (2 images $\times$ 600 fingers), and the testing set consists of 2,400 images (4 images $\times$ 600 fingers). Similarly, for database B, there are

(a) (b)

Fig. 9: Sample results from database C. (a) Original image; (b) Normalized gray image image from (a).

TABLE II: Time cost (h) of AG-NAS on three databases

Approach	Database A	Database B	Database C
w/o supernet (one epoch)	255.62	111.13	255.62
w/o supernet (5 epochs)	1227.84	555.58	1277.84
AG-NAS	5.12	2.21	5.12

1,050 images (5 images $\times$ 210 fingers) in the training set, 420 images (2 images $\times$ 210 fingers) in the validation set, and 1,050 images (5 images $\times$ 210 fingers) in the testing set. Finally, for database C, the training set comprises 2,400 images (4 images $\times$ 600 fingers), the validation set consists of 1,200 images (2 images $\times$ 600 fingers), and the testing set includes 2,400 images (4 images $\times$ 600 fingers).

In the experiments, we employ the SGD optimizer on the validation set to optimize the controller with a learning rate of 0.05, a momentum parameter of 0.9, and a weight decay of 2.5×10^{-4}. In addition, the Adam optimizer on the training set is used to optimize the supernet parameters with a learning rate of 0.001. During the training stage, the supernet is trained based on 10 batches of training data at each epoch, and the controller undergoes 50 times per epoch. For the training of each controller, 10 architecture descriptors are generated to determine 10 subnets, and the average accuracy of these subnets on the validation set is utilized to update controller's parameters. AG-NAS produces an optimal CNN after 200 training epochs, which is subsequently fine-tuned for an additional 1,000 epochs on the training set with a batch size of 32. The resulting CNN, exhibiting the best performance on the validation set, is saved and then evaluated on the testing set.

C. Search Efficiency

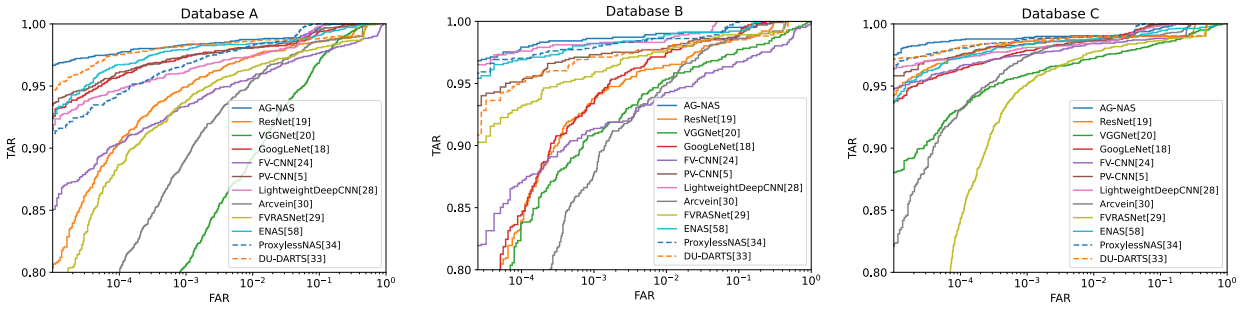
As shown in Fig. 4, the supernet includes 10 hidden layers, each offering 13 optional architectures, resulting in a search space of 13^{10} subnets. To efficiently explore this space, AG-NAS leverages reinforcement learning for network architecture search, consisting of a controller and a supernet iteratively trained to discover the optimal vein recognition architecture. To reduce time and storage costs, weights are shared among subnets in the supernet. During supernet training, the controller generates 10 subnets from the supernet, each trained on the training sets. This process, effectively training the supernet 10 times is repeated. For controller training, the controller generates 10 architecture descriptors, yielding 10 subnets from the supernet. The average accuracy of these 10 subnets on the validation set serves as a reward to update controller's parameters. The controller is iteratively trained 50 times, and this process alternates between controller and supernet training over 200 iterations. As a result, the optimal architecture is found after 10^5 (200 iterations $\times$ 50 times $\times$ 10 subnets) attempts by the controller, and 2,000 (200 iterations $\times$ 10 times) iterations of supernet training. If the supernet strategy is not employed, the controller outputs 10 descriptors, each used to obtain 10 subnets. These subnets are then individually trained from scratch on the training set for each iteration, with their average accuracy on the validation set serving as the reward for controller training. Without the supernet strategy, the controller makes 10^5 (200 iterations $\times$ 50 times $\times$ 10 subnets) attempts for architecture search, resulting in training 10^5 subnets or 10^5 iterations for subnets training. Comparing to the two approaches, our supernet mechanism (training only 2,000 subnets) incurs a significantly lower computation cost. Table II lists the time cost of our approach with and without the supernet for 10^5 controller attempts. From Table II, we observe that our approach with the supernet takes only a few GPU hours, i.e., 5.11 hours for database A, 2.22 hours for database B, and 5.11 hours for database C, to search for an optimal architecture from the 13^{10} search space. In contrast, our approach without the supernet strategy takes over 100 GPU hours on each database to determine the optimal architecture, even with each subnet trained for only one epoch. The time cost further increases significantly, surpassing 500 GPU hours for the three databases when each subnet is trained for 5 epochs. Typically, deep learning networks require more than one epoch for training. In addition, we list the time cost of representative approaches on the three databases in Table III. The experimental results in Table III illustrate that our AG-NAS approach achieves the lowest time costs on the three datasets. The following facts may explain such a good performance: AG-NAS employs the parameter sharing scheme to train different subnets, effectively reducing thereby parameter updating costs. Moreover, its architecture search for subnets occurs within a discrete search space. In contrast, ProxylessNAS [34] and DU-DARTS [33] optimize the architecture searching within a continuous space, leading to an increase in time costs. Overall, both ENAS [58] and AG-NAS exhibit comparable search times due to their utilization of parameter-sharing strategies during architecture search. The reduced computation time can be attributed to the efficacy of our attention mechanism, which is proficient in learning features with long-range dependencies and exploring optimal architectures within a reduced number of iteration epochs.

TABLE III: Time cost (h) of representative approaches on the three databases

Approach	Database A	Database B	Database C
ENAS [58]	5.48	2.26	5.94
ProxylessNAS [34]	9.94	3.21	8.04
DU-DARTS [33]	10.69	3.15	8.56
AG-NAS	5.12	2.21	5.12

D. Verification Performance

To evaluate the performance of AG-NAS, we conduct a comprehensive comparison with representative neural network models, including ResNet [19], VGGNet [20], GoogLeNet [18], FV-CNN [24], PV-CNN [5], LightweightDeepCNN [28],



(a) (b) (c)

Fig. 10: ROC of representative approaches on the test datasets. (a) Database A; (b) Database B; (c) Database C.

TABLE IV: EER (%) of representative approaches on the test datasets

Approach		Database A	Database B	Database C	Average
Deep learning-based	ResNet [19]	2.25	2.48	1.04	1.92
	VGGNet [20]	5.24	3.37	2.42	3.68
	GoogLeNet [18]	1.83	1.98	1.25	1.69
	FV-CNN [24]	3.21	4.16	1.58	2.98
	PV-CNN [5]	1.75	1.78	1.38	1.64
	LightweightDeepCNN [28]	2.21	1.28	1.46	1.65
	Arcvein [30]	2.92	2.78	1.21	2.30
	FVRASNet [29]	2.83	1.98	1.92	2.24
NAS-based	ENAS [58]	1.54	1.09	1.21	1.28
	ProxylessNAS [34]	1.59	1.39	1.04	1.34
	DU-DARTS [33]	1.37	1.78	1.08	1.41
	AG-NAS	**1.25**	**0.99**	**1.00**	**1.08**

Arcvein [30], and FVRASNet [29] as well as ENAS [58], DU-DARTS [33], and ProxylessNAS [34], on three finger-vein databases. All models undergo training on the designated training sets, and their performance on the respective testing sets is reported for comparative analysis. As described in Section IV-B, the testing sets consist of 2,400 images for database A, 1,050 images for database B, and 2,400 images for database C. In this experiment, we employ the commonly used equal error rate (EER) to estimate the performance of our approach. The EER is the point at which the false acceptance rate (FAR) equals the false rejection rate (FRR). Subsequently, the ROC curves are plotted with FAR against FRR. Table IV presents the EER values of representative approaches on the testing datasets of the three databases, and the corresponding ROC curves are illustrated in Fig. 10.

The experimental results reveal that the proposed AG-NAS outperforms the representative approaches, demonstrating the lowest verification errors of 1.25%, 0.99%, and 1.00% on databases A, B, and C, respectively. Furthermore, our approach achieves a lower average EER of 1.08% compared to other methods. In addition, NAS-based approaches consistently achieve significantly lower average EERs than deep learning-based approaches, although the EER might be slightly higher for a particular single database, showcasing global optimal performance rather than local optima. For example, while DU-DARTS exhibits a higher ERR than LightweightDeepCNN on database B (1.78% against 1.28%), it outperforms LightweightDeepCNN in improving the average EER (1.41% against 1.65%).

E. Identification Performance

In this experiment, we assess the identification performance of representative models on the three databases. The models, including ResNet, VGGNet, GoogLeNet, FV-CNN, PV-CNN, LightweightDeepCNN, Arcvein, FVRASNet, ENAS, ProxylessNAS, and DU-DARTS, are trained on the respective training sets and evaluated on the testing sets. The identification results for the three databases are summarized in Table V.

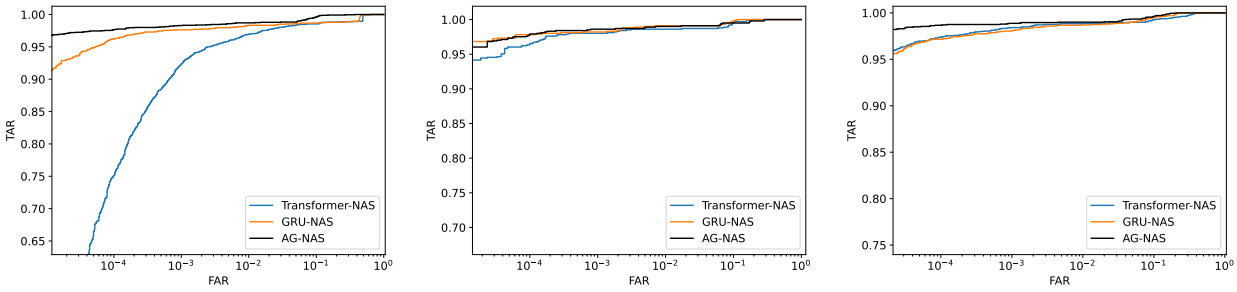
The consistent trends across the experimental results in Table V demonstrate that AG-NAS achieves the highest identification accuracy among the representative approaches, i.e., 97.33% for database A, 97.92% for database B, and 98.29% for database C. Overall, our approach exhibits more than a 1% improvement in the average identification accuracy. Furthermore, NAS-based approaches, i.e., ENAS, ProxylessNAS, and DU-DARTS, outperform most handcrafted architecture-based methods in terms of identification accuracy.

F. Discussions

The experimental results presented in Table II, Table IV, Table V, and Fig. 10 indicate that AG-NAS outperforms existing NAS-based approaches, including ENAS, ProxylessNAS, and DU-DARTS, as well as the manually designed CNNs, such as ResNet, VGGNet, GoogLeNet, FV-CNN, PV-CNN, LightweightDeepCNN, Arcvein, and FVRASNet. The superior performance can be attributed to several factors. The differentiable architecture search approaches, such as ProxylessNAS and DU-DARTS, suffer from two main issues: the weak robustness to the performance collapse and the limited generalization ability in the searched architectures,

TABLE V: Accuracy (%) of representative approaches on the test datasets

Approach		Database A	Database B	Database C	Average
Deep learning-based	ResNet [19]	96.21	93.76	98.25	96.07
	VGGNet [20]	91.21	92.57	96.04	93.27
	GoogLeNet [18]	96.21	95.84	97.04	96.36
	FV-CNN [24]	93.42	92.47	96.21	94.03
	PV-CNN [5]	96.25	96.53	97.17	96.65
	LightweightDeepCNN [28]	95.08	97.22	96.96	96.42
	Arcvein [30]	92.58	94.15	97.71	94.81
	FVRASNet [29]	94.13	95.24	96.75	95.37
NAS-based	ENAS [58]	96.13	97.03	96.58	96.58
	ProxylessNAS [34]	94.38	97.42	97.92	96.57
	DU-DARTS [33]	96.29	96.73	97.75	96.92
	AG-NAS	**97.33**	**97.92**	**98.29**	**97.85**



(a) (b) (c)

Fig. 11: Ablation experiments on the test datasets. (a) Database A; (b) Database B; (c) Database C.

resulting in poor performance [76]. Moreover, ENAS with an LSTM controller achieves limited performance because the traditional LSTM may fail to capture long-dependency features. In contrast, the attention mechanism in the transformer operates over longer distances and possesses the advantage of capturing long-dependencies within tokens. Therefore, the attention GRU module in AG-NAS demonstrates the capability to model longer sequences than conventional LSTM and GRU models, thereby enhancing architecture search capacity.

Overall, NAS-based models exhibit better performance than manually designed CNNs on the three databases, which may be attributed to the following facts: 1) The architectures of manually designed CNNs are shaped by *prior* human knowledge. In general, human experts face challenges in fully exploring the entire design space of DNN architectures due to their limited knowledge and experience, particularly when dealing with complex neural architectures. Furthermore, CNN manually designed for specific tasks may exhibit poorly for all vein recognition tasks or databases, resulting in poor generalization ability. In contrast, NAS-based approaches leverage rich prior knowledge by training the search strategy on the validation set. This empowers NAS to effectively search for an optimal architecture within an expansive space of subnets; 2) NAS automatically explores the architecture of deep neural networks through representation learning, establishing an objective connection to enhanced recognition performance. In existing NAS implementations, the controller takes a noise sequence as input and iteratively determines parameters in each layer to minimize decision errors in vein classification. AG-NAS mitigates the need for manual architecture design, eliminating the risk of discarding optimal networks for classification. Similar advantages have been substantiated in recent works across various tasks [32–34, 36]; 3) NAS-based approaches exhibit the capacity to automatically generate an optimal network for different vein recognition tasks. This capability enhances generalization and concurrently reduces the expenses associated with manual architecture design.

Our proposed AG-NAS demonstrates the ability to search for the optimal network architecture on different databases. Through experiments, we observe that the final networks are prone to employ separable convo-relu-norm with kernel of 3×3, max pool, and shortcut. For example, the final network for the first database comprises 9 layers, derived from a supernet with 13 layers via a direct path. This result shows that our approach can determine the number of layers for a given database, indicating that deeper networks may not necessarily yield superior performance. In addition, the consistent inclusion of shortcut connection (i.e., residual connection) in all hidden layers underscores their significant role in enhancing network performance, as supported by work [19]. Moreover, separable convo-relu-norm with kernel of 3×3 is determined to extract compact feature representation and reduced parameters. The proposed AG-NAS approach can extends beyond vein recognition, as it can be applied to other fields, such as face recognition and fingerprint recognition. Minor adaptations may be required by adjusting supernet's hyperparameters, such as the kernel size and number of kernels, to optimize performance based on variations in input image sizes across different tasks.

G. Ablation Experiment

In AG-NAS, the integration of the self-attention mechanism and GRU module enables the learning of dependencies among

TABLE VI: EER (%) of representative approaches on three databases

Approach	Database A	Database B	Database C	Average
Transformer-NAS	2.25	1.39	1.17	1.60
GRU-NAS	1.67	0.89	1.29	1.28
AG-NAS	1.25	0.99	1.00	1.08

TABLE VII: Accuracy (%) of representative approaches on three databases

Approach	Database A	Database B	Database C	Average
Transformer-NAS	95.83	97.13	97.04	96.67
GRU-NAS	96.71	97.62	96.58	96.97
AG-NAS	97.33	97.92	98.29	97.85

sequences in a recurrent manner. In this section, we conduct ablation experiments to evaluate the performance of each module w.r.t the improvement of recognition accuracy. To facilitate the description, we directly employ the transformer with two self-attention layers as a controller for architecture search, represented as transformer-NAS. In addition, we remove the self-attention mechanism from AG-NAS, denoted as GRU-NAS. The experimental results of the representative approaches on the three databases are listed in Tables VI and VII, and the corresponding ROC curves are depicted in Fig. 11, which demonstrate that the integration of the self-attention and GRU model allows to achieve higher recognition performance. In addition, we observe that GRU-NAS outperforms transformer-NAS, and this could be attributed to several facts. The transformer with extensive hyperparameters is typically capable of learning long-dependencies when trained on large training datasets. In our experiment, however, the limited training data may result in overfitting fo the transformer. Conversely, GRU is adept to capture temporal dependencies without requiring extensive data, thereby achieving higher recognition accuracy in our constrained dataset. Compared to transformer-NAS and GRU-NAS, AG-NAS can learn long-range dependencies with limited training data by introducing the attention mechanism into GRU. In other words, AG-NAS combines the advantages of GRU-NAS and transformer-NAS to optimize performance.

V. Conclusion

In this paper, we have proposed AG-NAS, a novel attention GRU-based neural architecture search scheme to search automatically for the optimal network architecture for finger-vein recognition. Our AG-NAS comprises two modules: the controller and the supernet. In AG-NAS, we investigated an attention GRU as the controller to generate the parameters based on which an optimal network is determined from the supernet. By introducing the attention mechanism into the recurrent network, we investigated an attention GRU to improve the feature representation capacity of learning long-range dependencies. We designed a parameter-sharing supernet to enhance search efficiency. In particular, AG-NAS could search for optimal network architectures for different datasets. For example, we obtained, for our dataset, a final network with 9 layers, including separable convo-relu-norm with a kernel of 3×3 and residual connection. These results imply that our approach can automatically infer an optimal lightweight network with residual connections, as such a lightweight network can alleviate overfitting on a small dataset while the residual connection can mitigate information loss and vanishing gradient. In our experiments, we compared AG-NAS with various vein classifiers and NAS-based approaches. The experimental results on three finger-vein databases demonstrate that our approach is capable of automatically searching for the optimal network architecture, yielding state-of-the-art performance. In the future, we plan to extend the application of our approach to other biometric modalities, such as palm vein, fingerprint, and hand vein, to improve the recognition performance.

References

[1] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge attacks on smartphone touch screens," in *Proceedings of the 4th USENIX Conference on Offensive Technologies (WOOT)*, 2010, pp. 1–7.

[2] S. Wiedenbeck, J. Waters, L. Sobrado, and J.-C. Birget, "Design and evaluation of a shoulder-surfing resistant graphical password scheme," in *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)*, 2006, pp. 177–184.

[3] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Proceedings of 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 1991, pp. 586–591.

[4] A. Jain, L. Hong, and R. Bolle, "On-line fingerprint verification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 302–314, 1997.

[5] H. Qin, M. A. El-Yacoubi, Y. Li, and C. Liu, "Multi-scale and multi-direction gan for cnn-based single palm-vein identification," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2652–2666, 2021.

[6] T. Tanaka and N. Kubo, "Biometric authentication by hand vein patterns," in *SICE 2004 Annual Conference*, vol. 1, 2004, pp. 249–253.

[7] H. Qin and M. A. El-Yacoubi, "Deep representation-based feature extraction and recovering for finger-vein verification," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1816–1829, 2017.

[8] J. Wang, G. Wang, and M. Zhou, "Bimodal vein data mining via cross-selected-domain knowledge transfer," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 733–74, 2017.

[9] I. S. Wang, H.-T. Chan, and C.-H. Hsia, "Finger-vein recognition using a nasnet with a cutout," in *Proceedings of 2021 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, 2021, pp. 1–2.

[10] Z. Yu, Y. Qin, X. Li, C. Zhao, Z. Lei, and G. Zhao, "Deep learning for face anti-spoofing: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 5, pp. 5609–5631, 2022.

[11] D. Menotti, G. Chiachia, A. Pinto, W. R. Schwartz, H. Pedrini, A. X. Falcao, and A. Rocha, "Deep representations for iris, face, and fingerprint spoofing detec-

tion," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 4, pp. 864–879, 2015.

[12] Y. Li, S. Ruan, H. Qin, S. Deng, and M. A. El-Yacoubi, "Transformer based defense gan against palm-vein adversarial attacks," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1509–1523, 2023.

[13] B. Huang, Y. Dai, R. Li, D. Tang, and W. Li, "Finger-vein authentication based on wide line detector and pattern normalization," in *Proceedings of 2010 International Conference on Pattern Recognition (ICPR)*, 2010, pp. 1269–1272.

[14] A. Kumar and Y. Zhou, "Human identification using finger images," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 2228–2244, 2012.

[15] E. C. Lee and K. R. Park, "Image restoration of skin scattering and optical blurring for finger vein recognition," *Optics and Lasers in Engineering*, vol. 49, no. 7, pp. 816–828, 2011.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *in NIPS*, pp. 1097–1105, 2012.

[17] W. Deng, H. Liu, J. Xu, H. Zhao, and Y. Song, "An improved quantum-inspired differential evolution algorithm for deep belief network," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 10, pp. 7319–7327, 2020.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[20] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *Proceedings of 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, pp. 730–734.

[21] H. Qin, M. A. El Yacoubi, J. Lin, and B. Liu, "An iterative deep neural network for hand-vein verification," *IEEE Access*, vol. 7, pp. 34 823–34 837, 2019.

[22] H. Qin and M. A. El Yacoubi, "End-to-end generative adversarial network for palm-vein recognition," in *Pattern Recognition and Artificial Intelligence*, Y. Lu, N. Vincent, P. C. Yuen, W.-S. Zheng, F. Cheriet, and C. Y. Suen, Eds. Cham: Springer International Publishing, 2020, pp. 714–724.

[23] W. Yang, C. Hui, Z. Chen, J.-H. Xue, and Q. Liao, "Fv-gan: Finger vein representation using generative adversarial networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 9, pp. 2512–2524, 2019.

[24] R. Das, E. Piciucco, E. Maiorana, and P. Campisi, "Convolutional neural network for finger-vein-based biometric identification," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 360–373, 2019.

[25] H. Qin, R. Hu, M. A. El-Yacoubi, Y. Li, and X. Gao, "Local attention transformer-based full-view finger-vein identification," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–16, 2022.

[26] J. Huang, W. Luo, W. Yang, A. Zheng, F. Lian, and W. Kang, "Fvt: finger vein transformer for authentication," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–13, 2022.

[27] B. Hou and R. Yan, "Convolutional autoencoder model for finger-vein verification," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 5, pp. 2067–2074, 2020.

[28] J. Shen, N. Liu, C. Xu, H. Sun, Y. Xiao, D. Li, and Y. Zhang, "Finger vein recognition algorithm based on lightweight deep convolutional neural network," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–13, 2022.

[29] W. Yang, W. Luo, W. Kang, Z. Huang, and Q. Wu, "Fvras-net: An embedded finger-vein recognition and antispoofing system using a unified cnn," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 11, pp. 8690–8701, 2020.

[30] B. Hou and R. Yan, "Arcvein-arccosine center loss for finger vein verification," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–11, 2021.

[31] H. Qin, C. Gong, Y. Li, X. Gao, and M. A. El-Yacoubi, "Label enhancement-based multiscale transformer for palm-vein recognition," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–17, 2023.

[32] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[33] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[34] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019.

[35] Z. Li, T. Xi, J. Deng, G. Zhang, S. Wen, and R. He, "Gp-nas: Gaussian process based neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 933–11 942.

[36] W. Jia, W. Xia, Y. Zhao, H. Min, and Y. X. Chen, "2d and 3d palmprint and palm vein recognition based on neural architecture search," *International Journal of Automation and Computing*, vol. 18, no. 3, p. 33, 2021.

[37] A. Mehrotra, A. G. C. Ramos, S. Bhattacharya, Ł. Dudziak, R. Vipperla, T. Chau, M. S. Abdelfattah, S. Ishtiaq, and N. D. Lane, "Nas-bench-asr: Reproducible neural architecture search for speech recognition," in *International Conference on Learning Representations*, 2021.

[38] W. Li, S. Wen, K. Shi, Y. Yang, and T. Huang, "Neural architecture search with a lightweight transformer for text-to-image synthesis," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 3, pp. 1567–1576,

2022.

[39] S. Deng, C. Fan, Y. Li, H. Qin, M. A. El-Yacoubi, and G. Zhou, "Gru-based neural architecture search for finger-vein identification," in *ICCCN*, 2023.

[40] N. Miura, A. Nagasaka, and Miyatake, "Feature extraction of finger-vein patterns based on repeated line tracking and its application to personal identification," *Machine Vision and Applications*, vol. 15, no. 4, pp. 194–203, 2004.

[41] L. Yang, G. Yang, Y. Yin, and X. Xi, "Finger vein recognition with anatomy structure analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1892–1905, 2017.

[42] H. Qin, X. He, X. Yao, and H. Li, "Finger-vein verification based on the curvature in radon space," *Expert Systems with Applications*, vol. 82, pp. 151–161, 2017.

[43] N. Miura, A. Nagasaka, and T. Miyatake, "Extraction of finger-vein patterns using maximum curvature points in image profiles," *IEICE TRANSACTIONS on Information and Systems*, vol. 90, no. 8, pp. 1185–1194, 2007.

[44] W. Song, T. Kim, H. C. Kim, J. H. Choi, H.-J. Kong, and S.-R. Lee, "A finger-vein verification system using mean curvature," *Pattern Recognition Letters*, vol. 32, no. 11, pp. 1541–1547, 2011.

[45] H. Qin, L. Qin, and C. Yu, "Region growth–based feature extraction method for finger-vein recognition," *Optical Engineering*, vol. 50, no. 5, pp. 057 208–057 208, 2011.

[46] H. Wang, M. Du, J. Zhou, and L. Tao, "ber local descriptors with variable curvature gabor filter for finger vein recognition," *IEEE Access*, vol. 7, pp. 108 261–108 277, 2019.

[47] B. Jun and D. Kim, "Robust face detection using local gradient patterns and evidence accumulation," *Pattern Recognition*, vol. 45, no. 9, pp. 3304–3316, 2012.

[48] B. Prommegger and A. Uhl, "Rotation invariant finger vein recognition," in *2019 IEEE 10th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, 2019, pp. 1–9.

[49] D. M. Sulaiman, A. M. Abdulazeez, H. Haron, and S. S. Sadiq, "Unsupervised learning approach-based new optimization k-means clustering for finger vein image localization," in *2019 International Conference on Advanced Science and Engineering (ICOASE)*. IEEE, 2019, pp. 82–87.

[50] K. Kapoor, S. Rani, M. Kumar, V. Chopra, and G. S. Brar, "Hybrid local phase quantization and grey wolf optimization based svm for finger vein recognition," *Multimedia Tools and Applications*, vol. 80, no. 10, pp. 15 233–15 271, 2021.

[51] J.-D. Wu and C.-T. Liu, "Finger-vein pattern identification using principal component analysis and the neural network technique," *Expert Systems with Applications*, vol. 38, no. 5, pp. 5423–5427, 2011.

[52] G. Yang, X. Xi, and Y. Yin, "Finger vein recognition based on (2d) 2 pca and metric learning," *Journal of Biomedicine and Biotechnology*, vol. 2012, pp. 1–9, 2012.

[53] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T. S. Huang, and S. Yan, "Sparse representation for computer vision and pattern recognition," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1031–1044, 2010.

[54] L. Yang, G. Yang, K. Wang, F. Hao, and Y. Yin, "Finger vein recognition via sparse reconstruction error constrained low-rank representation," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4869–4881, 2021.

[55] H. Qin and M. A. El-Yacoubi, "Deep representation for finger-vein image-quality assessment," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1677–1693, 2017.

[56] ——, "Finger-vein quality assessment based on deep features from grayscale and binary images," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 33, no. 11, p. 1940022, Oct. 2019.

[57] ——, "Finger-vein quality assessment by representation learning from binary images," in *Neural Information Processing*. Springer International Publishing, 2015, pp. 421–431.

[58] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*. PMLR, 2018, pp. 4095–4104.

[59] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, A. Filippov, and E. Burnaev, "Nas-bench-nlp: neural architecture search benchmark for natural language processing," *IEEE Access*, vol. 10, pp. 45 736–45 747, 2022.

[60] J.-H. Lee, J.-H. Chang, J.-M. Yang, and H.-G. Moon, "Nas-tasnet: Neural architecture search for time-domain speech separation," *IEEE Access*, vol. 10, pp. 56 031–56 043, 2022.

[61] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[62] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[63] J. Chang, Y. Guo, G. Meng, S. Xiang, C. Pan *et al.*, "Data: Differentiable architecture approximation," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[64] Y. Jiang, C. Hu, T. Xiao, C. Zhang, and J. Zhu, "Improved differentiable architecture search for language modeling and named entity recognition," in *Proceedings of EMNLP-IJCNLP*, 2019, pp. 3585–3590.

[65] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, "Block-wisely supervised neural architecture search with knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1989–1998.

[66] M. Zhang, H. Li, S. Pan, X. Chang, and S. Su, "Overcoming multi-model forgetting in one-shot nas with diversity maximization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*,

2020, pp. 7809–7818.
[67] S. Lu, Y. Hu, L. Yang, Z. Sun, J. Mei, Y. Zeng, X. Li, and T. ADlab, "Du-darts: Decreasing the uncertainty of differentiable architecture search," in *The 32nd British Machine Vision Conference. Virtual: British Machine Vision Association*, 2021.
[68] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7588–7598.
[69] Y. Kim, Y. Li, H. Park, Y. Venkatesha, and P. Panda, "Neural architecture search for spiking neural networks," in *European Conference on Computer Vision*. Springer, 2022, pp. 36–56.
[70] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
[71] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.
[72] X. Si, Z. Tian, X. Li, Z. Chen, G. Li, and J. D. Dormer, "Multi-step segmentation for prostate mr image based on reinforcement learning," in *Medical Imaging 2020: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 11315, 2020, p. 113152R.
[73] "http://www.corespirit.cn/."
[74] (2023) Finger-vein database. [Online]. Available: [Available: https://pan.baidu.com/s/1cjxPPPU2a0z4LDwX8GQfjQ
[75] Y. Lu, S. J. Xie, S. Yoon, J. Yang, and D. S. Park, "Robust finger vein roi localization based on flexible segmentation," *Sensors*, vol. 13, no. 11, pp. 14 339–14 366, 2013.
[76] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," *arXiv preprint arXiv:1909.09656*, 2019.

Huafeng Qin received BSc degree from the School of Mathematics and Physics and MEng degree from the College of Electronic and Automation from Chongqing University of Technology, and the PhD degree from the College of Opto-Electronic Engineering, Chongqing University. He was a visiting student for 12 months with Nanyang Technological University, Singapore, and then a postdoctoral researcher for two years with Université Paris-saclay, France. Currently, he is a professor with the School of Computer Science and Information Engineering, Chongqing Technology and Business University, China. His research interests include Biometrics (e.g., vein, face, and gait) and machine learning.

Chao Fan received the Bachelor's degree from the School of Computer Science and Information Engineering, Chongqing Technology and Business University, China, in June 2015. He is pursuing the Master's degree in the College of Computer Science, Chongqing University. His research interests include palm-vein identification and machine learning.

Shaojiang Deng received the Ph.D. degree from the College of Computer Science, Chongqing University, Chongqing, China, in June 2005. He is a full professor with the College of Computer Science, Chongqing University. His research interests include information security, mobile computing, and wireless sensor networks.

Yantao Li (Senior Member, IEEE) received the PhD degree from the College of Computer Science, Chongqing University, Chongqing, China, in December 2012. He is a professor with the College of Computer Science, Chongqing University. His research area includes mobile computing and security, Internet of things, sensor networks, and ubiquitous computing. He received the 2020 Best Paper Award from IEEE Internet Computing in 2022. He was a recipient of the Outstanding Ph.D. Thesis Award in Chongqing in 2014, and the Outstanding Master's Thesis Award in Chongqing in 2011. He serves as an Associate Editor for IEEE Internet of Things Journal (IoT-J).

Mounim A. El-Yacoubi (Member, IEEE) received the PhD degree from the University of Rennes, France, in 1996. He was with the Service de Recherche Technique de la Poste (SRTP) with Nantes, France, from 1992 to 1996. From 2001 to 2008, he was a senior software engineer with Parascript, Boulder (Colorado, USA), a world leader company in automatic processing of handwritten and printed documents (mail, checks, forms), for which he developed real-life software for address and check recognition. Since June 2008, he has been a Professor with Telecom SudParis, University of Paris Saclay. His main interests include machine learning, human gesture and activity recognition, human robot interaction, video surveillance and biometrics, information retrieval, and handwriting analysis and recognition.

Gang Zhou (Fellow, IEEE) received the PhD degree from the University of Virginia, in 2007. He is a full professor in the Computer Science Department at William & Mary. He was the graduate program director from 2015 to 2017. He is an IEEE Fellow, a co-Editor-In-Chief of ACM Transactions on Computing for Healthcare, and co-Area-Editor of IEEE Internet of Things Journal. His research interests include wearables & sensor systems, smart health, internet of things, wireless, ubiquitous & mobile computing, and security. In the past, he also served as an Associate Editor of ACM Transactions on Sensor Networks (TOSN), Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT), ACM Transactions on Computing for Healthcare (HEALTH), IEEE Internet of Things Journal (IoT-J), Elsevier Smart Health, Elsevier Computer Networks, and Springer Journal of Computer Science and Technology. He serves as a Steering Committee member (2018-present), General Chair (2019), and TPC Chair (2018 and 2023) of CHASE – ACM/IEEE's premier conference on Connected Health: Applications, Systems and Engineering Technologies. He received CAREER Award from the Computer and Information Science and Engineering Directory of the National Science Foundation (NSF) in 2013. He received 2020 Best Paper Award from IEEE Internet Computing. He received the Best Paper Award of 2010 IEEE International Conference on Network Protocols (ICNP). He received 2015 Plumeri Award for Faculty Excellence at William & Mary. He received ACM Recognition of Service Award from ACM SIG Governing Board in 2020. He also received an award for his outstanding service to the IEEE Instrumentation and Measurement Society in 2008.