A Novel Sleep Stage Classification Using CNN Generated by an Efficient Neural Architecture Search with a New Data Processing Trick

Yu Xue∗
School of Computer and Software
Nanjing University of Information Science and Technology
China
xueyu@nuist.edu.cn

Ziming Yuan
School of Computer and Software
Nanjing University of Information Science and Technology
China
YZM@nuist.edu.cn

Adam Slowik
Koszalin University of Technology
Poland
adam.slowik@tu.koszalin.pl

Abstract

With the development of automatic sleep stage classification (ASSC) techniques, many classical methods such as k-means, decision tree, and SVM have been used in automatic sleep stage classification. However, few methods explore deep learning on ASSC. Meanwhile, most deep learning methods require extensive expertise and suffer from a mass of handcrafted steps which are time-consuming especially when dealing with multi-classification tasks. In this paper, we propose an efficient five-sleep-stage classification method using convolutional neural networks (CNNs) with a novel data processing trick and we design neural architecture search (NAS) technique based on genetic algorithm (GA), NAS-G, to search for the best CNN architecture. Firstly, we attach each kernel with an adaptive coefficient to enhance the signal processing of the inputs. This can enhance the propagation of informative features and suppress the propagation of useless features in the early stage of the network. Then, we make full use of GA’s heuristic search and the advantage of no need for the gradient to search for the best architecture of CNN. Firstly, we attach each kernel with an adaptive coefficient to enhance the signal processing of the inputs. This can enhance the propagation of informative features and suppress the propagation of useless features in the early stage of the network. Then, we make full use of GA’s heuristic search and the advantage of no need for the gradient to search for the best architecture of CNN. This can achieve a CNN with better performance than a handcrafted one in a large search space at the minimum cost. We verify the convergence of our data processing trick and compare the performance of traditional CNNs before and after using our trick. Meanwhile, we compare the performance between the CNN generated through NAS-G and the traditional CNNs with our trick. The experiments demonstrate that the convergence of CNNs with data processing trick is faster than without data processing trick and the CNN with data processing trick generated by NAS-G outperforms the handcrafted counterparts that use the data processing trick too. Our research suggests that deep learning has a great potential on the electroencephalogram signal processing especially with the intensification of NAS. Meanwhile, NAS can exert greater power in practical engineering applications.

1. Introduction

Sleep [1] plays a significant role in human health and can reflect what diseases the body has such as narcolepsy [2], bruxism [3], obstructive sleep apnea [4], and etc. Therefore, researchers use sleep as an indicator of health [5] and confirm that the sleep stage can help people understand what’s going on in the body. In recent years, many researchers have been focusing on sleep analysis, especially automatic sleep stage classification (ASSC) [6].

Lots of ASSC methods have been proposed to support the sleep analysis based on the sleep standard of rechtschaffen and kales (R&K) or the american academy of sleep medicine (AASM) [7]. Most of them involve the two main steps including feature extraction and classification. Traditional machine learning methods are the most common analysis ways used by researchers to implement the two steps including feature extraction and classification. For example, [8] proposed wavelet transformation (WT) and fuzzy clustering based on fuzzy c-means algorithm (FCM) as the method to classify six sleep stages based on R&K from electroencephalogram (EEG). WT is an effective way of signal processing and is applied as EEG feature extraction. FCM is utilized as the classi-
fication. [9] took advantage of k-means clustering-based feature weighting to preprocess the data and then classified the data into six sleep stages using k-nearest neighbor and decision tree classifier. [10] designed Butterworth band-pass filters to filter and decompose EEG into five bands: delta, theta, alpha, beta, and gamma, then used support vector machine to classify EEG into two stages: awake (W) and stage 1 of sleep. [11] used just one channel of EEG signals which was extracted by tunable-Q wavelet transformation, then exploited bootstrap aggregating as a classifier. [12] combined structural graph similarity and the k-means to identify six sleep stages. [13] developed likelihood ratio decision tree classifier and extracted features from EEG, electromyogram (EMG), and electrooculogram (EOG) signals. Although those traditional machine learning methods are widely used, they must separate feature extraction and classification, and design their methods separately. This increases the complexity of designing an algorithm, and the quality of extracted features directly affects the performance of the classifier. Deep learning (DL) can effectively solve this problem because DL integrates feature extraction and classification. Meanwhile, the excellent representation ability of DL models can extract features that are more effective than traditional machine learning. [14] used eight channels of EEG signals and classified data into two stages with a multilayer perceptron feedforward neural network which is a classical application of DL in sleep classification. However, a mass of hyper-parameters needs to be considered, which is time-consuming. For example, when designing a feedforward neural network, many hyper-parameters need to be taken into consideration, such as the number of layers, the number of neurons in each layer, batch size of data, learning rate, etc. Therefore, the DL method is not explored adequately in sleep classification. Besides, DL cannot deal with signals from the human body, because it is mainly used to process data like images and the filters are designed to extract the local information. Different from channel information of images, signals are independent of each other which indicates that the information contained in each signal does not change with the change of other signals.

In this paper, we rethink the potential of DL in processing signals, and to reduce the cost of evaluating hyper-parameters when designing a model and further strengthen the ability of DL to deal with a practical problem, we introduce neural architecture search (NAS) to generate DL model. NAS is a powerful technique that can better combine DL theory and practice [15]. Specifically, we propose to design a special convolutional neural network (CNN) with an efficient data processing trick to extract features of signals, to process the signal more efficiently. More importantly, considering plenty of hyper-parameters that play a pivotal role in the performance of CNNs, we give full play to the classical branch of evolutionary computation, genetic algorithm’s (GA) heuristic search and the advantage of no gradient to design the efficient NAS, NAS-G, to produce the best architecture for our CNN.

We summary our contributions as follows:

1) We propose an efficient data processing trick in inputs to strengthen feature extraction. This is the key to combine CNN and signal processing. This trick can improve the performance of CNN on sleep classification.

2) Instead of designing CNN manually, we design the CNN architecture by using NAS, and we design our NAS with GA. The intensification of NAS not only saves the time consumption caused by human design but also takes into consideration the hyper-parameters that cannot be involved artificially.

2. Background

2.1. Deep representation mechanism of CNNs

In the last decade, CNNs have been developed rapidly and widely used in many fields such as pattern recognition [16] and image recognition [17] because of their powerful capability of representation. CNNs promote feature extraction from the manual design stage to the self-learning stage, so it can better fit the data [18]. The depth of CNN is an important factor. [19] proposed that a reasonable increase in the number of layers can effectively improve performance, which provides a design specification for subsequent research. However, the growth of layers increases the number of parameters and causes the device to be overwhelmed. Hence, many researchers improve the convolution operator to reduce the number of parameters without losing performance. [20] proposed to fuse various operations including convolution and skip connection. This is a device-friendly method because its fusion reduces a lot of parameters caused by skip connections. Similarly, [21] proposed to coalesce convolution, batch normalization [22], [23] proposed to use the divide-and-conquer strategy to compact convolution operator which forms a plug-and-play module.

2.2. Neural architecture search

There are many hyper-parameters including connection, depth, and size of filters in CNN to fine-tune. NAS [24] automates such manual operations, reduces the reliance on the expertise of the CNN architecture, and focuses on datasets. NAS makes it more convenient to exploit CNNs in practice. Currently, the performance of the network searched by NAS on some tasks has far exceeded the manually designed network on many tasks such as semantic segmentation [25], objective detection [26], and image classification [27]. With
the development of NAS, the framework of NAS can be divided into three dimensions, including search space, search strategy, and performance estimation strategy [28].

Search space parameterizes the architecture of CNNs and defines what kind of architectures can be discovered in principle. The simplest search space is chain-structured neural networks which are in line with the original CNN design mode that the connections just exist between adjacent layers [28]. Motivated by handcrafted architectures like [29], [30] proposed multi-branch networks based on cells and blocks, which incorporate skip connection that allows the architectures more complex and accurate.

Search strategy specifies how to explore search space encompassing Bayesian optimization, random search, reinforcement learning, evolutionary methods, and gradient-based methods. Evolutionary computation (EC) is the earliest method applied to NAS [31, 32]. [33] incorporated Bayesian optimization into NAS in 2013 and achieved state-of-the-art vision architectures.

The performance estimation strategy refers to the evaluation of the networks generated by the search strategy. The simplest operation is to train the network and evaluate it from scratch in a traditional way which is unfortunately computationally expensive. Lower fidelity estimate [34] reduces time cost through training with fewer epochs on the subset of data. Learning curve extrapolation [35] extrapolates the performance of architectures after just a few training epochs. Instead of training from scratch, Weight inheritance [36, 37] proposed that architectures inherit their parents’ weights. Weight sharing [38] just trains a one-shot model whose weights are then shared with the different architecture in the subgraph.

3. Proposed methods

3.1. Data processing trick

Every sample has ten signals (channels) including E1-M2, E2-M2, Chin1-Chin2, F4-M1, C4-M1, O2-M1, F3-M2, C3-M2, O1-M2, and RIP ECG (the details is described in Section IV). According to SENet [39] and CBAM [40], different features have different values and importance which means feature maps require further processing before they are used. Therefore, it is important to highlight informative features and suppress less useful ones. The purpose of the squeeze and excitation in SENet and the channel attention module in CBAM is to achieve a channel descriptor, and the spatial attention module in CBAM is to achieve the spatial descriptor. Differing from the image, the signal is highly serialized (up to 15360 in length), it’s hard to extract one figure to represent the distribution or statistics of a signal. As for spatial attention, as mentioned above, there is no relevance between channels. Hence, it is not reasonable to introduce spatial attention. To strengthen the propagation of features, we attach a coefficient that can be learned to each kernel in the input layer, emphasize the informative features, suppress the less useful ones, and enhance the propagation of features. Figure 1 demonstrates how the coefficients work. This gives the convolutional filters a certain degree of autonomous selectivity to adaptively evaluate the importance of each signal, so that the features which have a greater impact on the classification result are better propagated, while the propagation of the less influential features is impaired. It is more flexible when extracting representations in the input layer.

Figure 1: To attach a coefficient \( (k_i) \) to each kernel \( (kernel_i) \) of the input layer.

3.2. Performing NAS

In consideration of a large number of hyper-parameters such as the number of layers, the type of operation performed on each layer, and the number of output channels in each layer, the process of exploring the best hyper-parameters suitable for the particular problem requires a lot of professional skills and empirical knowledge. Besides, constant experimentation, test, and troubleshooting are needed too. To mitigate complexity, we incorporate NAS based on the GA, NAS-G. The framework of NAS-G is shown in Algorithm 1.

First, initialize a population of \( N \), and calculate the fitness value of the individuals in the population (line 1-2). Second, in the \( T \) generations, each generation selects two parent individuals from the parent population through the binary tournament selection [41] according to the fitness. Then perform crossover and mutation operations on the two parent individuals to generate offspring individuals and form the offspring population (line 3-10). Evaluate the fitness of offspring (line 11). Finally, a new population is selected from the parent and offspring population according to fitness through environmental selection (line 12). When the \( T \) generations terminate, the best CNN will be decoded from the best individual that has the maximal fitness.

To introduce our NAS-G more clearly, we elucidate its mechanism from the following three aspects: encoding strategy, fitness evaluation, and offspring generation.
Algorithm 1 NAS-G algorithm

Input: The population size $N$, the maximal generation number $T$, the crossover probability $\mu$, the mutation probability $\nu$.

1: $P_0 \leftarrow$ Initialize $N$ architectures as a population using encoding strategy;
2: Train and validate each individual, then take the highest accuracy as the fitness of each individual in $P_0$;
3: $t \leftarrow 0$;
4: while $t < T$ do
5: \hspace{1em} $Q_t \leftarrow \emptyset$;
6: \hspace{2em} while $|Q_t| < N$ do
7: \hspace{3em} $p_1, p_2 \leftarrow$ Select two architectures from $P_t$ with binary tournament selection [41];
8: \hspace{3em} $q_1, q_2 \leftarrow$ Generate two architectures by $q_1$ and $q_2$ by crossover operation with the probability $\mu$, and mutation operation with the probability $\nu$;
9: \hspace{3em} $Q_t \leftarrow Q_t \cup q_1 \cup q_2$;
10: \hspace{2em} end while
11: \hspace{1em} Train and evaluate architectures’ performance in $Q_t$;
12: \hspace{1em} $P_{t+1} \leftarrow$ Select $N$ architectures from $P_t \cup Q_t$ by environmental selection;
13: \hspace{1em} $t \leftarrow t + 1$;
14: end while

Output: The architecture of a CNN with the best performance.

1) Encoding strategy. To improve the performance of architecture in search space and accelerate the speed of searching, we introduce the prior knowledge of two skip blocks, ResNet block (RB) [29] and DenseNet block (DB) [42]. In addition to these blocks, a pooling block is employed too, which just contains a pooling layer. According to the experience of orthodox hand-designed CNN, we use the filter of (1,3), and the stride is set to 1. To change the length of the individual more flexibly, we design three units, ResNet unit (RU), DenseNet unit (DU), and Pooling unit (PU), to contain different convolutional components. RU contains ResNet blocks, DU contains DenseNet blocks, and PU contains one pooling block, respectively. Figure 2 shows the details of RB, DB, RU, and DU. Hence, one CNN is built based on the three units. A list is created to denote an individual (architecture), the length of which is defined randomly within a range, then generate multiple RUs, DUs, and PUs in the list randomly. Therefore, an individual represents the list of units. Note that the number of PUs cannot exceed $\log_2 N$, where $N$ denotes the length of the input. For example, if the size of the input is (1,100), the number of PUs is 6 at most, otherwise, it leads to logical error. As a result, the parameters of a RU are unit tag, input channel, output channel, and the number of ResNet blocks. The parameters of DU are unit tag, input channel, output channel, the number of DenseNet blocks, and the growth rate $k$. The parameter of PU is the unit tag and the type of pooling (max pooling or average pooling). Figure 3 shows the details of the encoding strategy of the three types of units. The $unit$ represents the type of unit and 0 denotes RU, 1 denotes DU, and 2 denotes PU. The $in$ and $out$ in RU and DU represent number of input channel and number of output channel, respectively. The $num$ in RU represents number of RB and the $num$ in DU represents the number of DU. The $k$ is growth rate. The $type$ in PU represents the type of pooling and 0 denotes max pooling and 1 denotes average pooling.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures.png}
\caption{Two prior knowledge contain RB (a) and DB (b). Multiple RBs constitute a RU (c) and multiple DBs constitute a DU (d).}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures.png}
\caption{Encoding strategy of the three type of unit.}
\end{figure}

2) Fitness evaluation. The fitness of an individual indicates how well it fits the data. According to the encoding strategy, we first decode each individual into a CNN, then we train the CNN on training data. Finally, we take the accuracy of each individual on the validation set as its fitness value. Equation 1 formulates the fitness. \( Acc(\bullet) \) measures the performance of a CNN denoted by $\Gamma$ on the validation dataset $D_v$ after trained on the training dataset $D_t$.

\[
fitness = Acc(\Gamma, D_t, D_v) \quad (1)
\]

3) Offspring generation. First of all, two parent architectures are selected. To produce the offspring architectures with better property than parents, it needs to be ensured that the performance of selected architectures is quite excellent, because the nature of GA is the inheritance of merit from parents, and at the same time, the trap of local optima needs to be avoided. To this end, binary tournament selection is used to achieve
Figure 4: Crossover process. Firstly, select one crossover point in each parent architecture. Different from traditional crossover, the selection of crossover points between architectures does not affect each other (a). Then, swap the second parts after the two crossover points to generate offspring architectures (b).

4. Experiments
4.1. Datasets

This goal. First, it selects two architectures in population, then the architecture with higher fitness is chosen as one parent. The other is selected in the same way.

Secondly, the crossover is performed on these two parent architectures. Considering the depth of CNN is variable and is one important factor influencing the quality of CNNs, the crossover of traditional GA conducted on individuals with the same length is not suitable for CNN design. Consequently, we choose one crossover point in each parent architecture randomly as Figure 4 (a) shows, then the first part of parent₁ connects to the second part of parent₂, and the first part of parent₂ connects to the second part of parent₁ as Figure 4 (b) shows. A red rectangle or blue rectangle in Figure 4 represents a unit that is DU, RU, or PU. Note that the first unit cannot be selected as crossover point, otherwise, empty solutions will be generated.

Finally, the mutation is conducted on the two offspring architectures. Three types including ‘add’, ‘reduce’ and ‘alter’ constitute mutation. ‘add’ means adding an extra unit RU, DU, or PU. Note that the number of PUs cannot exceed the maximum. ‘reduce’ means reducing a unit. ‘alter’ means altering the attribute of a unit, such as the output channel or the number of ResNet blocks or DenseNet blocks.

Table 1: The number of samples for each stage.

| Sleep stages | W  | N1 | N2 | N3 | REM |
|--------------|----|----|----|----|-----|
| Number       | 386| 949| 255| 746| 553 |
Table 2: Six comparison models ($E_0$- $E_5$) and our model $E_6$

| Models | Description |
|--------|-------------|
| $E_0$  | Do not make any process on the original signals (10, 15360, 1) and perform convolution operation on it with the filter of (1,3). |
| $E_1$  | Transform the shape of (10, 15360, 1) into (200, 256, 3), pad with 0 to form the shape of (256,256,3) like the way dealing with images and perform convolution operation on it with the filter of (3,3). |
| $E_2$  | Transform the shape of (10, 15360, 1) into (128, 120, 10), pad with 0 to form the shape of (128,128,10) and perform convolution operation on it with the filter of (3,3). |
| $E_3$  | Transform each signal into (128, 120), concatenate them together as (128, 120, 10) and perform convolution operation on it with the filter of (3,3). |
| $E_4$  | Bases on $E_3$, attach a coefficient to each kernel in the input layer. |
| $E_5$  | Transform each signal into a one-dimensional matrix and the style of filters is (1,$X$). |
| $E_6$  | Based on $E_5$, attach a coefficient to each kernel in the input layer. |

4.2. Training details

4.2.1 Data processing trick

To verify the performance of our methods, we design seven models which are not very complicated. Table 2 shows the details of the seven models. In $E_0$, we keep signals in their original state and then process them, which means we concatenate them into one channel and perform a convolution operation with the filter of (3,3) on the result. In the second model $E_1$, inspired by processing traditional pictures, we transform the shape of original signal data (10, 15360) into (256, 200, 3) because pictures have three channels. In $E_2$, given the ten signals, we reshape the shape to (128, 120, 10) which has ten channels like the original signals. Both $E_1$ and $E_2$ do not consider the independence between each signal and just regard the entire ten signals as one 2-D sample. So in $E_3$, we transform the shape of each signal into (128, 120) and concatenate them together as (128, 120, 10). Based on $E_3$, we attach a coefficient to each kernel in the input layer in $E_4$. In every model above, we design each filter as (3, 3) due to their shape. In $E_5$, we incorporate the 1-D convolution. We first convert the signals into 1-D matrices, (1, 15360, 10), at the same time, as the number of the network increases, we design the convolution filters as (1, 30), (1, 20), (1, 10), (1, 5), and (3, 1). The reason why we set the size of the filter to progressively decrease is that the data size is quite large in the early stage in the network, especially when the signal data is just input (1, 15360, 10). The size of (1, 30) is better to extract the features because large-size filters can mine the features in long serialized data more efficiently. Then as the feature size decreases, we also reduce the size of the filters. After the convolution using the filters with the size style (1,$X$), we rebuild the results into a 2-D style and utilize the filter with the size of (3, 1) as Figure 9 shows. The purpose of this process is to extract the features between channels. Based on $E_5$, we attach a coefficient to each kernel in the input layer in $E_6$. And for clarity, we just give the architectures in $E_4$ and $E_6$ as Figure 10 and Figure 11 demonstrate. The training setting in each...
model on each dataset is the same: the batch size is 64, the epoch is 80, the optimizer is Adam, and the learning rate is 0.0001. We set the batch size to 64 because those architectures are not very complicated (compared with the searched ones), and this value depends on our computing resource.

4.2.2 NAS-G

Different from the architecture of CNN used in $E_6$, we add the component of RB in addition to DB to improve the networks’ ability to extract the features when using NAS-G to search architectures. Meanwhile, considering the relatively high complexity of the architecture to be searched, we introduce the pooling operation and cancel the convolution filter (3, 1).

Figure 8: The signals from E1-M2 to RIP ECG.

Figure 9: (a) $C$ features extracted by filter of (1, X), and each feature is an independent channel. (b) One new feature concatenated by previous $C$ features and the filter of (3, 1) conducted on this feature.

Given one GPU (NVIDIA 2080Ti) under usage, we set population size and the generation number to 17 and 20 re-
4.2.3 Traditional models

We compare our method with three models of the ResNet family [29] and two classical models with attention mechanism, SENet [39] and CBAM [40]. To better extract features of signals, we just modify their filter of (3, 3) with (1, 3). The training setting of each model is that batch size is 64, the epoch is 80, the optimizer is Adam, and the learning rate is 0.0001. To exhibit the performance of our method, we first train and validate the five models with their vanilla architecture on the six datasets, then we train and validate them with our data processing trick.

5. Results and discussion

5.1. Results of data processing trick

Figure 12 shows the accuracy on six validation datasets when training the seven models (E₀ - E₆) where the horizontal axis denotes the training steps and the vertical axis denotes the classification accuracy. Each curve represents the accuracy trend of one model on one validation dataset. On the right side of each subgraph is the average accuracy of each model after 1500 steps. All models converge around 1300 steps. Before convergence, E₀, E₁, E₂, and E₃ start to fit data from accuracy of 0.2, E₅ starts to fit data from accuracy of 0.3, and E₄ and E₆ start to fit the data from an accuracy of 0.4. After convergence, E₁ has the worst performance of around 0.3, and is far from others. E₆ has the highest performance around 0.7. The performance of E₅, E₄, and E₀ is very close, but experiments still show that the performance of E₅ is better than that of E₄ whose performance is better than that of E₀’s performance. Although E₀ has not been improved in any way, its performance is not worse than that of E₁, E₂, and E₃.

We can infer from the figure that E₁ has the worst performance because E₁ completely disrupts the internal correlation in the processing of raw data. E₀ does not change each raw single signal when merges them into one channel. E₂ breaks raw data too, but compared to E₁, the signals are not very disturbed which results in the improvement in accuracy. Differing from the two models above, E₃ first transforms each signal into two-dimensional and then concatenates them along channels, which destroys every signal (one-dimensional) in the original data, but maintains the independence of each signal, so the validation performance is improved. E₄ adds coefficients on the top of E₃, and filters with coefficients strengthen the feature extraction and propagation, which means that the so it is obvious that the performance is better than that of E₃. Besides, we can deduce that breaking the inner correlation is not wise, and this is why E₂, E₃, and E₄ can not outperform E₀, but the coefficients can make up for this shortcoming very well, which can be inferred from the comparison between E₃ and E₄. In particular, we can see from the comparison between E₁, ...
$E_2$, and $E_3$ that the less damage is done to original data, the better performance the CNN gets. Although $E_5$ does not use the data processing trick, $E_5$ completely keeps original data, so before the convergence, it can fit the data better than $E_0$-$E_3$. From the comparison between $E_5$ and $E_0$-$E_3$, we can draw that keeping raw data can perfect the fitting. On basis of $E_5$, $E_6$ adds coefficients, and we can see that the fitting ability before convergence surpasses the previous models, except for $E_4$ and surpasses all models after convergence. Interestingly, $E_4$ and $E_6$ are the only models that use the data processing trick. With the enhancement of this trick, their ability to fit in the early stage is better than other models, and their accuracy after convergence is also quite high among all models. $E_6$’ accuracy is the highest, and the accuracy of $E_4$ is second only to that of $E_5$. The main reason why the performance of $E_6$ exceeds that of $E_4$ is that $E_6$ keeps the integrity of signals. Meanwhile, $E_5$ keeps the integrity of signals but does not use the trick and $E_4$ uses the trick but destroys the integrity of signals, but their accuracy is very close. So we argue that data processing trick can make up for the performance loss caused by signal damage.

To sum up, we can conclude that 1) architecture damage is inadvisable for signal classification using CNNs. 2) the coefficients can strengthen the ability of fitting (keeping integrity of the signals also involves this capacity), and at the same time make up for the performance loss caused by signals’ damage. In most cases, our algorithm performs better on common CNNs, because our model not only uses each signal as a complete unit for feature extraction, but the coefficients can strengthen the extraction of features.

5.2. Result of NAS-G

After 20 generations, a total of 340 individuals (architectures) are generated. On account of the characteristic of GA that in every generation, crossover and mutation can generate some individuals that are the same as the previous generations did, in our experiment, there are 59 repeats and in the statistical process. For clarity, they are ignored. As a result, there are 281 individuals that are exhibited below. Figure 13 (a) shows the result of NAS-G, where the horizontal axis denotes the individuals, and the vertical axis denotes the fitness (accuracy or performance). As we can see, the fitness values of individuals present two completely different parts, and one is around 0.8 and the other is zero. Meanwhile, fitness values appear most intensively in the first 50 generations. The improvement of fitness values after the 200th generation is more obvious than before it. To elucidate the process of NAS-G more clearly, we give the tendency of fitness in which individuals with a fitness of zero are abandoned as Figure 13 (b) illustrates. The blue line represents the tendency and value in the blue line is the mean value of the neighborhood ($\zeta - 35, \zeta + 35$) where $\zeta$ denotes the original value (the red one). In the front and the rear, we copy the front values and the rear values to meet the length of the neighborhood. We can infer that with the increase of generation, individuals with good performance are constantly produced. The fitness values are between 0.75 and 0.90 and the main fitness is between 0.80 to 0.90. The fitness value is
rising in the first 100 generations, the plain period appears between 100th and 150th generation, and the fitness value is rising again after 150th generations. Figure 14 shows the perfect architecture searched by NAS whose fitness (accuracy) is 89.6%. This architecture involves two RUs, three DUs, and eleven PUs.

In the initial stage, most individuals have a fitness value of zero, because, in the early stages of evolution, the architectures are too complex to be supported by computing power, which makes performance evaluation impossible. However, with the evolution of the population and the crossover and mutation between different individuals, too complex architectures are gradually eliminated, so in the subsequent evolution process, most individuals are adapted to computing power. This confirms the powerful search ability of GA. In addition to the elimination of overly complex individuals, the performance is constantly evolving. Under the mechanism of environmental selection, crossover allows good architecture components (hyper-parameters) to be passed on to the next generations. Therefore, in the early stages of evolution, the fitness values of individuals are rising. However, in the mid-term, the fitness values have a short period of saturation, and the population performance falls into the plain stage. This is because the populations enter the stage of the local optimal solution. This is mainly due to the effect of crossover because crossover just combines components in different architectures and this combination can find a locally optimal solution in a short time which indicates that crossover’s local search capability is very strong. However, new components can be generated through mutation, which makes new architectures different from others, thus the population can jump out of the local optimal, which indicates that the global search ability of mutation is very strong. This is also the reason why population performance rebounds again in the later stage of evolution. In this architecture generated by NAS-G, the proportion of PUs is the largest and the PUs are used to reduce the data dimension, which, therefore, indicates that there is a lot of noise in the dataset and a denoising work is required, and this work is done by the PUs. When designing a model artificially, it is difficult to consider superimposing multiple PUs because most researchers think that continuous PUs reduce information and are detrimental to model performance. This also confirms that NAS-G not only saves a lot of time overhead due to artificial design, but also searches for architectures that humans cannot consider because of the huge search space.

5.3. Result of comparison with traditional models

As shown in Table 3, when the data processing trick is not used, the performance of ResNet-50 is better than ResNet-18 on those datasets except Dataset 0, but the difference between the two is not very large. The performance of ResNet-101 is the worst on those datasets except Dataset 0. The performance of SENet and CBAM surpasses the ResNet family. SENet is the best among the 5 models on those datasets except Dataset 0. After using the data processing trick, all models are improved. ResNet-50 still exceeds ResNet-18 and ResNet-101 on datasets except Dataset 0 and ResNet-101 remains the worst. SENet and CBAM still are superior to the ResNet family. It is worth noting that the performance of the model generated through NAS-G is the best.

We can infer from Table 3 that on those datasets except Dataset 0, the performance of ResNet-50 in the ResNet family is the best, and ResNet-101 is the worst. This shows that the architecture of ResNet-18 is too simple to fit the data well. ResNet-50 has twice as many parameters as ResNet-18, so it can extract more information and fit data better. The amount of parameters of Reset-101 is twice that of ResNet-50, but the worst effect is obtained. This is because the architecture of ResNet-101 is too complex, and the performance degrades during the training. This indicates that for our datasets, ResNet-50’s architecture complexity is the most suitable, and increasing the complexity on this basis may result in a decrease in performance. SENet and CBAM achieve better performance among the five models because to some extent, they are affected by the attention mechanism and capture the meaningful information in signals. The architecture searched by NAS-G achieves the best performance. Interestingly, these models’ accuracy Different from the handcrafted architectures, NAS-G generates 281 architectures. In the process of evolution, individuals of different lengths are generated through the crossover. These individuals contain the different number of parameters and hyper-parameters, including the size of the input and output layers, the pooling style, etc., not only screening models suitable for computing power, but also keeping the better hyper-parameters. At the same time, mutation can generate new hyper-parameters, expand the architecture space, jump out of the local optimal solution, search for the global optimal solution, and further search for higher-performance architectures. We argue that the models generated through NAS-G are the best architecture.

6. Conclusions

This paper reconsiders and explores the potential of DL on sleep classification by employing a data processing trick in CNNs, and further develops a NAS method using GA to enhance the performance of DL. We find that the integrity of the original signal is an important factor affecting classification accuracy. The more complete the signals, the higher the classification accuracy. It is worth noting that our trick makes up for the performance loss caused by the damage of signals. Besides, our data processing trick can improve the accuracy of current traditional models on sleep classi-
Figure 13: Results of NAS-G. (a) is the result of all individuals without repeats in 20 generations and the bottom red dots in the figure represent the overly complex individuals. (b) is the result of individuals without overly complex individuals.

Figure 14: The architecture searched by NAS-G.

Table 3: Comparison with traditional models.

| model   | Dataset 0 | Dataset 1 | Dataset 2 | Dataset 3 | Dataset 4 | Dataset 5 |
|---------|-----------|-----------|-----------|-----------|-----------|-----------|
|         | vanilla (%) | +trick (%) | vanilla (%) | +trick (%) | vanilla (%) | +trick (%) | vanilla (%) | +trick (%) | vanilla (%) | +trick (%) |
| ResNet-18 | 83.5     | 85.5     | 83.7     | 85.6     | 85.8     | 86.5     | 85.5     | 85.8     | 87.0     | 87.5     | 87.7     | 88.1     |
| ResNet-50 | 85.6     | 86.4     | 83.9     | 86.4     | 85.5     | 86.4     | 86.3     | 86.6     | 87.6     | 88.0     | 87.9     | 88.3     |
| ResNet-101 | 85.9    | 86.1     | 82.7     | 84.3     | 84.8     | 85.6     | 85.1     | 85.3     | 86.4     | 87.0     | 86.3     | 87.3     |
| SENet   | 87.2     | 87.9     | 84.3     | 86.8     | 86.3     | 87.0     | 88.0     | 88.6     | 88.0     | 88.6     | 88.3     | 88.8     |
| CBAM    | 87.5     | 88.1     | 84.2     | 86.1     | 86.3     | 87.2     | 86.6     | 86.9     | 87.9     | 88.8     | 88.2     | 89.1     |
| NAS-G(ours) | -       | 89.6     | -        | 88.8     | -        | 89.1     | -        | 87.7     | -        | 89.4     | -        | 89.3     |
fication. More importantly, NAS-G, as the tool of deep learning, can intensify the performance of DL in the field of sleep classification, because compared with human design, NAS-G saves a lot of time. At the same time, based on the huge search space, NAS-G searches for architectures that humans cannot consider. We hope that our method can promote the application of DL in sleep classification or signal processing, and our work can encourage researchers in different fields to pay attention to NAS, because NAS is a strong method that can merge different fields and DL.

References

[1] Bing Zhai, Ignacio Perez-Pozuelo, Emma AD Clifton, Joao Palotti, and Yu Guan. Making sense of sleep: Multimodal sleep stage classification in a large, diverse population using movement and cardiac sensing. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 4(2):1–33, 2020. doi:10.1145/3444829.

[2] Lucie Barateau and Yves Dauvilliers. Narcolepsy, Idiopathic Hypersomnia, and Dysautonomia. Autonomic Nervous System and Sleep: Order and Disorder, pages 187–198, 2021. doi:10.1007/978-3-030-62263-3_17.

[3] Md Belal Bin Heyat, Dakun Lai, Faijan Akhtar, Mohd Ammar Bin Hayat, Shafan Azad, Shadab Azad, and Shajan Azad. Bruxism Detection Using Single-Channel C4-A1 on Human Sleep S2 Stage Recording. Intelligent Data Analysis: From Data Gathering to Data Comprehension, pages 347–367, 2020. doi:10.1007/9781107518447-17.

[4] Daniel J Gottlieb and Naresh M Punjabi. Diagnosis and management of obstructive sleep apnea: a review. Jama, 323(14):1389–1400, 2020. doi:10.1001/jama.2020.3514.

[5] Joanna Garstang, Marta Cohen, Edwin A Mitchell, and Peter Sidebotham. Classification of sleep-related sudden unexpected death in infancy: A national survey. Acta Paediatrica, 110(3):869–874, 2021. doi:10.1111/apa.15472.

[6] Sirvan Khalighi, Teresa Sousa, Jose Moutinho Santos, and Urbano Nunes. ISRUC-Sleep: a comprehensive public dataset for sleep researchers. Computer Methods and Programs in Biomedicine, 124:180–192, 2016. doi:10.1016/j.cmpb.2015.10.013.

[7] Conrad Iber, Sonia Ancoli-Israel, Andrew L Chessson, Stuart F Quan, et al. The AASM manual for the scoring of sleep and associated events: rules, terminology and technical specifications, volume 1. American academy of sleep medicine Westchester, IL, 2007.

[8] M. Obayya and F. E. Z. Abou-Chadi. Automatic classification of sleep stages using EEG records based on Fuzzy c-means (FCM) algorithm. In 31st National Radio Science Conference, pages 265–272, 2014. doi:10.1109/NRSC.2014.6835085.

[9] Salih Gunes, Kemal Polat, and Sebnem Yosunkaya. Efficient sleep stage recognition system based on EEG signal using k-means clustering based feature weighting. Expert Systems with Applications, 37(12):7922–7928, 2010. doi:10.1016/j.eswa.2010.04.043.

[10] Khalid AI Aboalayon, Helen T Ocbagabir, and Miad Faezipour. Efficient sleep stage classification based on EEG signals. In IEEE Long Island Systems, Applications and Technology Conference 2014, pages 1–6, 2014. doi:10.1109/LISAT.2014.6845193.

[11] Ahnaf Rashik Hassan and Abdulhamit Subasi. A decision support system for automated identification of sleep stages from single-channel EEG signals. Knowledge-Based Systems, 128:115–124, 2017. doi:10.1016/j.knosys.2017.05.005.

[12] Mohammed Diykh, Yan Li, and Peng Wen. EEG sleep stages classification based on time domain features and structural graph similarity. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 24(11):1159–1168, 2016. doi:10.1109/TNSRE.2016.2552539.

[13] Kristin M Gunnarsdottir, Charlene E Gamaldo, Rachel ME Salas, Joshua B Ewen, Richard P Allen, and Sridevi V Sarma. A novel sleep stage scoring system: combining expert-based rules with a decision tree classifier. In 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pages 3240–3243, 2018. doi:10.1109/EMBC.2018.8513039.

[14] N Sriraam, TK Padma Shri, and Uma Maheshwari. Recognition of wake-sleep stage 1 multichannel EEG patterns using spectral entropy features for drowsiness detection. Australasian Physical & Engineering Sciences in Medicine, 39(3):797–806, 2016. doi:10.1007/s13246-016-0472-8.

[15] Yu-Chao Gu, Li-Juan Wang, Yun Liu, Yi Yang, Yu-Huan Wu, Shao-Ping Lu, and Ming-Ming Cheng. Dots: Decoupling operation and topology in differentiable architecture search. In Proceedings of the
[16] Rudi Kurniawan Samsuryadi and Fatma Susilawati Mohamad. Automated handwriting analysis based on pattern recognition: A survey. Indonesian Journal of Electrical Engineering and Computer Science, 22(1):196–206, 2021. doi:10.11591/ijeecs.v22.i1.pp196-206.

[17] Xiwu Zhang, Lei Wang, and Yan Su. Visual place recognition: A survey from deep learning perspective. Pattern Recognition, 113:107760, 2021. doi:10.1016/j.patcog.2020.107760.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998. doi:10.1109/5.726791.

[19] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In International Conference on Learning Representations, 2015.

[20] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 13733–13742, 2021.

[21] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse Branch Block: Building a Convolution as an Inception-like Unit. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10886–10895, 2021.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning, pages 448–456, 2015.

[23] Chen Zhang, Yinghao Xu, and Yujun Shen. CompConv: A Compact Convolution Module for Efficient Feature Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3012–3021, 2021.

[24] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12965–12974, 2020.

[25] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021. doi:10.1109/TPAMI.2021.3059968.

[26] Xinyu Huang, Dongming Chen, Tao Ren, and Dongqi Wang. A survey of community detection methods in multilayer networks. Data Mining and Knowledge Discovery, 35(1):1–45, 2021. doi:10.1007/s10618-020-00716-6.

[27] Sen Jia, Shuguo Jiang, Zhijie Lin, Nanying Li, Meng Xu, and Shiqi Yu. A survey: Deep learning for hyperspectral image classification with few labeled samples. Neurocomputing, 448:179–204, 2021. doi:10.1016/j.neucom.2021.03.035.

[28] Hangyu Zhu, Haoyu Zhang, and Yaochu Jin. From federated learning to federated neural architecture search: a survey. Complex & Intelligent Systems, 7(2):639–657, 2021. doi:10.1007/s40747-020-00247-z.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

[30] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 8697–8710, 2018.

[31] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks, 5(1):54–65, 1994. doi:10.1109/72.265960.

[32] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2):99–127, 2002. doi:10.1162/106365602320169811.

[33] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In International Conference on Machine Learning, pages 115–123, 2013.

[34] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A
novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research, 18(1):6765–6816, 2017. 3

[35] Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum dependent learning curves in kernel regression and wide neural networks. In International Conference on Machine Learning, pages 1024–1034, 2020. 3

[36] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In European Conference on Computer Vision, pages 35–51, 2020. doi:10.1007/978-3-030-58452-8_3.

[37] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In European Conference on Computer Vision, pages 544–560, 2020. doi:10.1007/978-3-030-58517-4_32. 3

[38] Liam Li and Ameet Talwalkar. Random Search and Reproducibility for Neural Architecture Search. In Ryan P. Adams and Vibhav Gogate, editors, Proceedings of The 35th Uncertainty in Artificial Intelligence Conference, volume 115, pages 367–377. PMLR, 22–25 Jul 2020. 3

[39] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-Excitation Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7132–7141, 2018. 3, 8

[40] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional Block Attention Module. In Proceedings of the European Conference on Computer Vision, pages 3–19, 2018. 3, 8

[41] Tobias Blickle and Lothar Thiele. A Comparison of Selection Schemes Used in Evolutionary Algorithms. Evolutionary Computation, 4(4):361–394, 1996. doi:10.1162/evco.1996.4.4.361. 3, 4

[42] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4700–4708, 2017. 4