NEURON HIERARCHICAL NETWORKS

Anonymous authors
Paper under double-blind review

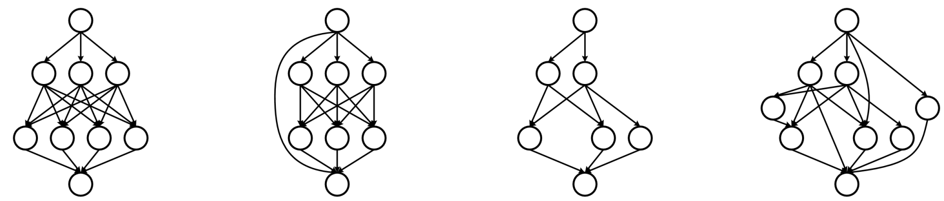
ABSTRACT

In this paper, we propose a neural network framework called *neuron hierarchical network* (NHN), that evolves beyond the hierarchy in layers, and concentrates on the hierarchy of neurons. We observe mass redundancy in the weights of both handcrafted and randomly searched architectures. Inspired by the development of human brains, we prune low-sensitivity neurons in the model and add new neurons to the graph, and the relation between individual neurons are emphasized and the existence of layers weakened. We propose a process to discover the best base model by random architecture search, and discover the best locations and connections of the added neurons by evolutionary search. Experiment results show that the NHN achieves higher test accuracy on Cifar-10 than state-of-the-art handcrafted and randomly searched architectures, while requiring much fewer parameters and less searching time.

1 INTRODUCTION

Neural networks can be designed either by human experts or search algorithms, both of which have gained great success in image classification and language modeling (Huang et al., 2017; Yang et al., 2018; Pham et al., 2018). Network architectures designed by both means are mostly layer-based or block-based, which means that the fundamental components are either operation layers or blocks that consist of several layers. A clear tendency can be observed that models with more parameters generally have better performances. It is a well-established fact that redundancy of parameters exists widely in handcrafted neural networks (Han et al., 2015b; Tai et al., 2015; Lin et al., 2017). We find that such claim holds for architectures discovered by random search or evolutionary search as well. The pruning of unimportant neurons emphasizes the hierarchical relation between individual neurons. Additionally, the decrease in accuracy after parameter reduction is generally inevitable. Therefore, we propose a heuristic procedure to construct neuron-based network architectures by pruning redundant connections and neurons in layer-based models and adding new neurons to strengthen the neuron hierarchy while achieving competitive performances as layer-hierarchy models. Experiments show that NHN achieves higher test accuracy than DenseNet (Huang et al., 2017), SMASH (Brock et al., 2018) and hierarchical representation (Liu et al., 2018) with much fewer parameters.

Handcrafted architectures. Successful convolutional neural networks (CNNs) designed by human experts can be sketchily categorized by the way data flow through the networks, i.e., plain networks and branching networks. A notable example of *plain* networks would be VGG nets (Simonyan and Zisserman, 2014), where there are only one input and output path in each hidden layer. However, in a *branching* network, the computation flow splits somewhere in the network and merges in a latter layer (Ahmed and Torresani, 2017). The splitting and aggregation may occur multiple times in a single network. Many have discovered numerous branching network architectures whose performances surpass plain ones while requiring fewer parameters. Skip connections (He et al., 2015; 2016a; Huang et al., 2017) are increasingly popular in improving the performance of deep neural networks, and it becomes common to observe additional convolutions (or other forms of operations) stacked between large layers (Lin et al., 2013; Szegedy et al., 2017; Chen et al., 2017). In fact, the ”stacked-between” operations can be considered as part of a generalized residual block. Multi-branching computation graphs benefit addressing the gradient vanishing problem during the gradient descent training (He et al., 2016a; Huang et al., 2017). The distinguished techniques mentioned above (plus more listed in Table 1) share the same idea of weakening the hierarchy between layers by introducing complex paths to the data flow. The idea is further highlighted by architecture search algorithms.



(a) Plain network. (b) Branching network. (c) Network pruning. (d) Neuron hierarchy.

Figure 1: Neuron hierarchical networks as a generalization of various network structures. Each node in the four graphs indicates an actual neuron in the network. NHN may be derived by performing network pruning *(c)* on a plain/branching network *(a,b)*, then adding new neurons to break the layer hierarchy *(d)*.

Random and evolutionary architectures. Machine learning algorithms evolve fast. Designing neural networks that perform remarkably on a given task requires ample experience. It has been found that neural networks are not only good at autonomically extracting useful features from raw data, but also capable of finding optimal network architectures to that end. Neural architecture search (NAS) (Zoph and Le, 2017; Zoph et al., 2017) has been attested to its ability to design network architectures for language modeling and image classification. However, candidate models have to be entirely randomly generated and fully trained, therefore NAS is extremely computation intensive, which dims its competitive performances to handcrafted architectures. Many efforts have been devoted to reducing the computational costs of NAS while ensuring sufficient capacity of search space (Liu et al., 2018; Zhong et al., 2018; Liu et al., 2017; Cai et al., 2017). Two major ideas to achieve the purpose are to design individual reusable cells rather than the entire networks or to share trained weights between models. Recently, Pham et al. (2018) proposed to describe each candidate model as a subgraph of a single directed acyclic graph (DAG). By sharing weights between submodels, the searching time of NAS is reduced by 1000X in term of GPU-hours. Genetic algorithms are also applied in searching the optimal architectures of CNNs (Miikkulainen et al., 2017; Xie and Yuille, 2017; Real et al., 2017). Real et al. (2018) proposed regularized evolution (RE) to remove the oldest models from the population, instead of removing the worst ones as in traditional tournament selection (Goldberg and Deb, 1991). The best CNN model discovered by RE achieves the state-of-the-art performance on Cifar-10, i.e., 2.13% test accuracy on average. However, the number of parameters it requires is as large as nearly 35 million. Convolutional neural fabrics (CNF) (Saxena and Verbeek, 2016; Veniat and Denoyer, 2017) and other forms of random search methods (Baker et al., 2017; Brock et al., 2018; Dong et al., 2018) have been investigated as well.

Layer-wise to neuron-wise hierarchy. Take the best model discovered by macro search in ENAS (Pham et al., 2018) for example. The 12-layer CNN model contains over 21 million parameters and achieves 4.23% test accuracy on Cifar-10. If we remove $75\% - 90\%$ parameters in all 3×3 and 5×5 convolutions, the test accuracy is hardly compromised after the same duration of re-training. Even though the architectures in all the search methods are described as directed graphs, each node in these graphs represents either an operation layer (e.g. convolution or pooling layer) or an operation block (e.g. residual block). None of the nodes stands for an actual individual neuron in the network. On one hand, random search and evolutionary search tend to discover architectures that contain complex branching paths. On the other hand, the pruned versions of such architectures work nearly as well as intact ones. These facts bring about the hypothesis that the hierarchy of neurons should work as well as the hierarchy of layers. Please note that we do not simply replace layers with individual neurons, considering layers are composed of abundant neurons. We need the sufficient number of neurons to meet the feature representation requirements. A good hierarchical architecture of neurons may be discovered by either random search or evolutionary search, or combined. The search process must be carefully designed.

In this paper, we propose a three-step course to discover the optimal neuron hierarchy for image classification (see Figure 1), i.e., (1) discover the optimal layer hierarchy with ENAS, (2) prune unimportant neurons in the discovered layer-wise model, and (3) randomly add new neurons to the pruned model to enrich the expressive capacity of neuron hierarchy networks. It is worth pointing out that the three-step procedure is also an imitation of the natural development of human brains

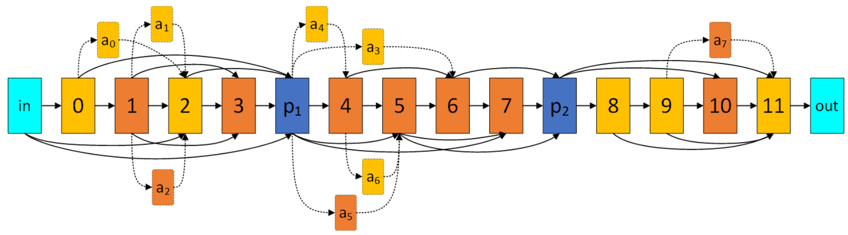
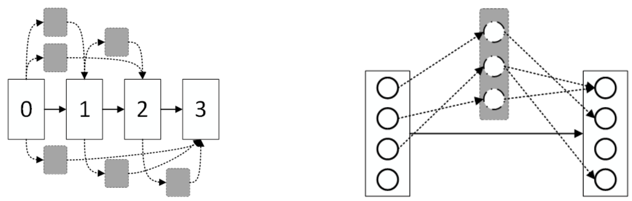


Figure 2: Illustration of discovered layer hierarchy and potential locations for add-on neurons. Block p denotes the average pooling cell. Block a denotes the location where add-on neurons may occur. The yellow and orange colored blocks represent 3×3 and 5×5 convolutions, respectively. Better viewed in color.



(a) Location search.

(b) Direction search.

Figure 3: Illustration of location-direction search (LDS). First, search for optimal locations where add-on neurons may occur *(a)*. Second, search for optimal direction at each given location *(b)* that add-on neurons point to the existing neurons. White blocks stand for existing layers and neurons. Gray blocks stand for add-on neurons.

(Craik and Bialystok, 2006). For example, the creation and searching by ENAS correspond to the mass neurogenesis before birth. The pruning of unimportant neurons corresponds to the apoptosis before puberty (Elmore, 2007). The addition of new neurons to the pruned model corresponds to the persisting neurogenesis during adulthood (Ming and Song, 2011). Although the existence of neurogenesis in mature brains is being debated in the field of neuroscience (Sorrells et al., 2018; Boldrini et al., 2018), the software simulation of such process by our work indicates that it is helpful in improving the learning capability of neural networks.

2 The proposed approach

In this section, we propose a heuristic algorithm to derive network architectures that emphasize the relation between individual neurons based on traditional layer-based models by network pruning and adding new neurons.

2.1 Discover the layer hierarchy

We directly employ the *macro* search method proposed by Pham et al. (2018) to discover a 12-layer CNN architecture for image classification. Note that we need to distinguish the important neurons from the less important ones, thus we are not performing Dropout (Srivastava et al., 2014) during the searching and re-training in ENAS. Even though Dropout has proved to be helpful in increasing the generalization power of networks, it distributes the feature extracting capabilities evenly in all neurons, which makes it hard for us to tell which neurons are more important than others. We will, however, employ Dropout in the final fine-tuning stage. We use the cosine schedule (Loshchilov and Hutter, 2017) that involves 4 warm restarts (for a total of 310 epochs) to adjust the learning rate during training, as in Pham et al. (2018), with the maximum value of 0.05 and minimum of 0.001. Cutout (DeVries and Taylor, 2017) is used to enrich the sample capacity and improve the generalization of the model. The architecture discovered by ENAS is illustrated in Figure 2 (the large blocks connected with solid lines)

2.2 NETWORK PRUNING

Network pruning (Han et al., 2015b;a; Lin et al., 2017) has been used to get rid of unimportant parameters in CNNs. When followed by quantization and sparsification techniques, the network pruning methods can considerably reduce the memory and bandwidth consumption of deep networks in distributed systems. But it requires pre-learning of the connectivity and a very long time of re-training to restore the performance. We employ the dynamic network pruning (DNP) method proposed by Anonymous (2018) to remove redundant weights. Specifically, during the search of optimal layer-hierarchy model, we replace the global average pooling (Lin et al., 2013) with grouped convolution (Xie et al., 2017; Kaiser et al., 2016), in which the group number equals the number of both input and output channels and the kernel sizes equal the sizes of input feature maps, so that the grouped convolution exactly maps each feature map to a single value. We also smooth out all 3×3 and 5×5 convolution kernels with 3×3 Gaussian filters after initialization. After training the optimal model discovered by ENAS from scratch, we find that only a small portion of trained kernels have apparent texture appearance, while a majority of them are practically flat. We remove nearly 85% of the kernels based on standard deviation thresholds. Accuracy drop is observed on the pruned model, it expresses the need for finding a way to improve the representation capability of our model, which leads to the idea of adding additional neurons to the network.

2.3 EVOLUTIONARY ADD-ON NEURONS

In order to distinguish the newly added neurons from the existing neurons, we refer to the former ones as *add-on neurons*. The add-on neurons are free to be inserted between two arbitrary neurons that are originally located in different layers, since a connection within a layer could cause confusion about computation order. The input and output neurons of an add-on neuron must share the same data dimension, so that we can define a standard rule to create add-on neurons. Take the best architecture discovered by ENAS for example. The 12-layer architecture is divided by two average pooling layers into three segments, with each containing 4 convolution cells. The data dimensions are uniformed in each segment, therefore add-on neurons are not allowed to be inserted across separate segments. Two types of operations are allowed in an add-on neuron, i.e., 3×3 convolution and 5×5 convolution.

Algorithm 1: Location-Direction Search (LDS)

Input : A directed acyclic graph $\mathcal{G}$, in which each node represents a layer in the layer-hierarchy network.
Output: $\tilde{\mathcal{G}} = \mathcal{G} + \mathcal{A}$, in which $\mathcal{A}$ represents the discovered add-on neurons and their connections to the input network $\mathcal{G}$.
Require: Population capacity K, cycle number T, train samples B_{train}, test samples B_{test}
<u>Location search</u>
Initialize: Population $\mathcal{P}_l = \{p_l^k, k = 1, 2, \ldots, K\}$, in which $p_l^k \leftarrow$ **GenerateLoc**($\mathcal{G}$)
for $t \leftarrow 1$ **to** T **do**
    foreach p_l^k **do**
        Train p_l^k with train sample batches $\{B_{train}^{(k-1)\cdot M+1}, \ldots, B_{train}^{k \cdot M}\}$
        Evaluate p_l^k with test sample batches $\{B_{test}^{(k-1)\cdot N+1}, \ldots, B_{test}^{k \cdot N}\}$
    end
    $\mathcal{P}_l \leftarrow$ **Sort**($\mathcal{P}_l$) by test accuracy decreasingly
    $\{p_l^{11}, \ldots, p_l^{15}\} \leftarrow$ **Recombine**($p_l^1, p_l^2, p_l^3, p_l^4$)
    for $i \leftarrow 1$ **to** 5 **do**
        $p_l^{i+15} \leftarrow$ **Mutate**(p_l^i)
    end
end
$\mathcal{G}_c \leftarrow$ best model from $\mathcal{P}_l$
<u>Direction search</u>
Population $\mathcal{P}_d = \{p_d\}$, $p_d \leftarrow$ **GenerateDir**($\mathcal{G}_c$)
return $\tilde{\mathcal{G}} \leftarrow p_d$

Location and direction search. In order to increase the search speed for add-on neurons, we propose a *location-direction search* (LDS) method to search the specific positions and connections independently, which means that we first search for the most possible locations where the add-on neurons may occur, then decide on their specific input and output connections. The *location* search is carried out in an evolutionary fashion, as described in Algorithm 1. First, we define an initial population ($GenerateLoc$). Each individual in the population stands for a possible combination of add-on locations. During each evolution cycle, we train every model in the population for several iterations (M) and we (1) crossover the best models ($Recombine$) to reproduce new models, (2) mutate the best models ($Mutate$) to produce new models, and (3) remove the worst models from the population. The crossover ($Recombine$) is performed as follows. Let $p_l^1, p_l^2, p_l^3, p_l^4$ denote the best 4 models, in which lower indices are assigned to better models. The crossover pairs to generate 5 new models are selected as $\{(p_l^1, p_l^2), (p_l^1, p_l^3), (p_l^1, p_l^4), (p_l^2, p_l^3), (p_l^2, p_l^4)\}$. After the best combination of add-on locations is discovered, an evolutionary search on add-on *directions* is performed. The direction of an add-on neuron (i.e. to choose which neurons to connect with) at given location is selected sparsely ($GenerateDir$), since we do not want add-on neurons at the same location to marginally form a new layer. Because the potential number of different combinations of add-on directions can be extremely large, we simply re-generate all combinations at each evolution cycle to find the best one. No mutation or crossover will be performed during direction search.

For example, let us employ LDS to the first segment in Figure 2, i.e., $layers$ $\{0, 1, 2, 3\}$, as separately shown in Figure 3a. We define the location where one or several add-on neurons may occur as a new sparse convolution layer (e.g., a_0 in Figure 2). All possible locations for add-on neurons are $\{(0, 1), (0, 2), (1, 2), (0, 3), (1, 3), (2, 3)\}$, in which tuple (a, b) means that add-on neurons are inserted between layer a and b, as illustrated as gray blocks in Figure 3a. If we enforce a restraint that only one input add-on location with the same kernel size is allowed for each existing layer and include the pooling layer into search space, the overall location search complexity will be 1.9×10^8 (which is $(4!)^3 \cdot (4!)^3$, since there are three segments in the network and we are using two different kernel sizes of add-on neurons). When the optimal add-on locations are determined, we perform the direction search with the sparsity of 0.1, which means that we will leave out 90% of all possible connections and leave the remaining effective neurons as the search results. Keeping neurons with high sensitivities is recommended, because we need the add-on neurons to improve the performance of the model. Therefore, the standard deviation thresholds are used to determine the optimal add-on directions.

Weight sharing in add-on search. We adopt the idea of sharing weights between submodels by Pham et al. (2018) to increase the search speed in LDS. Specifically, we define all possible connections at all possible locations beforehand. Let $\mathcal{W} = \{w_{l,d} | l \in \mathcal{L}, d \in \mathcal{D}\}$ denote all shared weights for LDS, in which $w_{l,d}$ denotes the weight at location l and direction d, and $\mathcal{L}$ and $\mathcal{D}$ stand for all possible locations and directions, respectively. When generating new subgraphs for location or direction search, we simply select the corresponding weights in $\mathcal{W}_{k,t} = \{w_{l,d} | l \in \mathcal{L}_{k,t}, d \in \mathcal{D}_{k,t}\}$, in which (k, t) stands for the kth model in the search population at evolution cycle t. In general, all $w_{l,d}$ in $\mathcal{W}$ are trained in an asynchronous fashion.

The proposed approach for neuron hierarchical networks can be described as Algorithm 2.

Algorithm 2: Construct a neuron hierarchical network (NHN).

Require: Number of layers L
Initialize: Layer-hierarchy network $\mathcal{G}$
Layer hierarchy search
  $\mathcal{G} \leftarrow$ **SmoothInit**($\mathcal{G}$)
  $\mathcal{G} \leftarrow$ **ENAS**($\mathcal{G}$)
Network pruning
  $\mathcal{G} \leftarrow$ **DNP**($\mathcal{G}$)
  Train $\mathcal{G}$ from scratch
Neuron hierarchy search
  $\tilde{\mathcal{G}} \leftarrow$ **LDS**($\mathcal{G}$)
  Train $\tilde{\mathcal{G}}$ from scratch
return $\tilde{\mathcal{G}}$

Table 1: Cifar-10 test errors of NHN and other methods. The first block represents handcrafted architectures. The second block represents architectures discovered by random search methods. The third block represents evolutionary architectures. Note that "d/o" stands for Dropout, "s/s" stands for Shake-Shake regulation, "c/o" stands for Cutout, and "d/p" stands for Drop-path. $Score = \sqrt{(e/2)^2 + (n/25)^2}$, where e denotes the test error in percentage and n denotes the number of parameters in million. Smaller score indicates better overall performance. **The performance of Evolving DNN on Cifar-10 is reported by Liu et al. (2018).*

Method	Parameters	Error	Score
ResNet-110 (He et al., 2016a)	1.7M	6.43%	3.22
pre-act-ResNet-1001 (He et al., 2016b)	10.2M	4.62%	2.35
WRN-40-10 + d/o (Zagoruyko and Komodakis, 2016)	56M	3.80%	2.94
WRN-28-20 + SGDR (Loshchilov and Hutter, 2017)	145.8M	3.74%	6.12
DenseNet (Huang et al., 2017)	27.2M	3.74%	2.16
ResNeXt (Xie et al., 2017)	68.1M	3.58%	3.26
DenseNet + s/s (Gastaldi, 2017)	26.2M	2.86%	**1.77**
CNF (Saxena and Verbeek, 2016)	21.2M	7.43%	3.81
MetaQNN (Baker et al., 2017)	11.2M	6.92%	3.49
Budgeted CNF (Veniat and Denoyer, 2017)	7.6M	5.12%	2.58
FractalNet + d/p + d/o (Larsson et al., 2017)	38.6M	4.60%	2.77
PPP-Net (Dong et al., 2018)	11.3M	4.36%	2.23
ENAS macro (Pham et al., 2018)	21.3M	4.23%	2.28
SMASH (Brock et al., 2018)	16.0M	4.03%	2.11
NAS + more filters (Zoph and Le, 2017)	37.4M	3.65%	2.36
Block-QNN-S + more filters (Zhong et al., 2018)	39.8M	3.54%	2.38
ENAS micro (Pham et al., 2018)	4.6M	3.54%	1.78
EAS by NT (DenseNet) (Cai et al., 2017)	10.7M	3.44%	1.77
Progressive NAS (Liu et al., 2017)	3.2M	3.41%	**1.71**
CL on ResNeXt (Ahmed and Torresani, 2017)	34.4M	3.17%	2.10
Genetic CNN (Xie and Yuille, 2017)	-	7.10%	-
Evolving DNN (Miikkulainen et al., 2017)*	-	7.30%	-
Large-scale evolution (Real et al., 2017)	-	5.40%	-
Evo. search on hier. repr. (Liu et al., 2018)	61.3M	3.63%	3.05
AmoebaNet + c/o (Real et al., 2018)	34.9M	2.13%	**1.76**
NHN	7.0M	3.72%	1.88
NHN + c/o	7.0M	3.48%	**1.76**

3 Experiments

We are using the Cifar-10 (Krizhevsky and Hinton, 2009) dataset to train and evaluate our neuron hierarchy network. The Cifar-10 dataset consists of 50k 32×32 3-channel training images in 10 classes, and 10k test images. The standard pre-processing is applied, i.e., subtracting the mean and dividing the standard deviation on each channel. The standard data augmentation is used for the training set, i.e., padding the images to 40×40 and randomly cropping down to 32×32, then flipping horizontally by a probability of 0.5. All experiments are run on a single NVIDIA GTX 1080Ti video card, which has sufficient GPU memory for the searching and training of the NHN.

Layer hierarchy search. We use the same hyperparameters as in Pham et al. (2018) to perform the macro search on the entire architecture, as explained in Section 2.1. It takes ENAS no more than 16 hours to find a layer-hierarchy model with test accuracy of 96.55% on Cifar-10 with Cutout. If trained without Cutout, the model's test accuracy is 95.08%. The re-training of the best model from scratch takes almost 9 hours.

Network pruning. We perform the DNP method to prune low sensitivity connections in the best model discovered by ENAS, as described in Section 2.2. We also clean up the network by removing all the neurons with zero input or output connection after DNP. We prune nearly 85% of input connections of all 3×3 and 5×5 convolution layers on average. In total, more than 70% of the parameters are pruned. The overall pruning ratio decreases because of the immense applications of 1×1 convolutions in ENAS, which are not directly removed by DNP. Evident impact on the performance by pruning is observed. We re-train the pruned network for another 160 epochs, which takes about 5 hours. The test accuracy is slightly dropped to 96.20% (with Cutout).

Neuron hierarchy search. The population capacity (K) for *location* search is set to 20. The individual model p_l^k is described as the gene code of length of 24, in which the first 12 genes indicate whether an add-on location exists with kernel size of 3×3 at each existing layer (since there are 3 segments and we need 4 genes for each segment), and the last 12 genes are for 5×5. A positive gene indicates there are add-on neurons connecting into this layer and the value denotes the input layer index. A negative gene value (-1) means that no add-on neuron is connected to this layer. Out of avoiding too many add-on neurons being added to the model, the gene value of -1 is generated by a probability of 0.8, and the rest of the values are sampled uniformly. Each model is trained for 25 iterations (M) with the batch size of 100, so that a complete training cycle of the population exactly traverses the entire training set of Cifar-10. Submodels are evaluated by single iteration ($N = 1$) with the batch size of 500. Considering the sparsity of add-on locations and that a slight change in the gene value could lead to dramatic changes in performance, we simply re-generate 5 new models instead of performing mutation on the best models. A cosine schedule for learning rate that gradually decreases from 0.05 to 0.001 is used. The location search takes around 10 hours (300 evolution cycles) to complete. The best 3 sets of discovered locations are illustrated in Figure 2 (the small blocks). It is apparent that having more add-on neurons at low-level layers works better for image classification. The *direction* search is performed as described in Section 2.3. The direction search takes about 9 hours (300 cycles) to finish. Only weights in add-on neurons are trained and weights in existing neurons are fixed during the LDS, because we are trying to find the add-on neurons that are most compatible with existing neurons. It also saves a great number of computation resources when the majority of parameters are not trained.

After the best locations and directions of add-on neurons have been discovered, we re-train the entire network from scratch. Cutout (DeVries and Taylor, 2017) and Dropout (Srivastava et al., 2014) are used for better generalization. The training schedule resembles the one used by Pham et al. (2018). Nesterov (Nesterov, 1983) is used as the gradient descent method, in which the momentum term is set to 0.9 and l_2 regularization of 2×10^{-4} is used. Gradient clipping (Pascanu et al., 2013) with the threshold of 5.0 is used to overcome the potential gradient explosion. In fact, most of the weights (i.e. the weights from the original layer-wise network) have undergone three passes of training, i.e., during ENAS, after DNP, and after LDS. It is reasonable to believe that the overall training has seen several warm restarts, hence the reason why no warm restart is introduced in the training stages after DNP and LDS. The final training takes 6 hours (260 epochs) to complete.

The final test accuracy of NHN is **96.52%**. The performance of NHN in comparison with other methods is listed in Table 1. The total number of parameters in the NHN is approximately **7M**. We can see that the test accuracy of NHN is better than most of the methods listed in Table 1. However, several methods achieve lower test errors than NHN, e.g. DenseNet with Shake-Shake, connectivity learning on ResNeXt and AmoebaNet with Cutout. However, the amounts of parameters of these methods are huge. In order to examine the overall performances of methods in terms of test errors and parameter numbers, we define the performance score as follows. $score = \sqrt{(e/2)^2 + (n/25)^2}$, where e denotes test error (%) and n denotes number of parameters (M). A lower score value indicates better performance of the model. We can see that except for Progressive NAS which has marginally better performance score, the NHN is highly competitive against all other methods in terms of low test error and small parameter number.

On the whole, the searching and training of the neuron hierarchy network take approximately **46 hours** in total, in which the search time of neuron hierarchy is 10 hours. The search time of NHN, however, is indeed more than that of ENAS, since we actually perform a complete ENAS process among other operations during the search. When comparing to NAS (Zoph and Le, 2017) (which takes 450 GPUs to search for 3-4 days), the speedup is as high as **700X-940X** and parameter number is reduced by 81%. NHN achieves higher accuracy than SMASH (Brock et al., 2018) with 56% fewer parameters. Moreover, it also has higher accuracy than hierarchical representation (Liu et al., 2018) with 89% fewer parameters and a speedup in search time of **152X**, when the latter costs 1.5

days on 200 GPUs to evaluate 7000 samples. Our approach achieves more than **5X** speedup with 35% fewer parameters compared to the EAS by NT (Cai et al., 2017), which costs 2 days on 5 GPUs to achieve similar accuracy. Compared to another method that achieves similar accuracy, Progressive NAS (Liu et al., 2017), NHN achieves speedup in GPU time of **50X**.

4 RELATED WORK AND DISCUSSION

Cai et al. (2017) proposed a network architecture search method which allows layer transformations during architecture search by enforcing Actor Networks for widening or deepening. Although it allows more neurons and parameters being added to the network during architecture search, the models it discovers are still layer-based, thus redundancy in weights cannot be overlooked. Ahmed and Torresani (2017) proposed to learn the connectivity in ResNeXt (Xie et al., 2017). They create a uniform grid on which each node represents a residual block, and connections are only allowed between neighboring rows of nodes, whereas in our work, each node represents an actual neuron, the network is not necessarily described as a grid, and connections are not under strict restrains. The performance of the best architecture discovered by connectivity learning is impressive, but the method only applies to ResNeXt.

We did not choose fixed or handcrafted architectures as the base model because we believe that experiments conducted on a randomly searched model would be more compelling. There are also pruning-related issues with fixed models, for example, performances of ResNets and DenseNet are extremely sensitive to pruning. The training from scratch on the pruned architecture is crucial, because without it, the model only has a test accuracy of 15.7%. The NHN is not built upon the results by ENAS micro search even though it presents higher test accuracy while requiring fewer parameters than macro search. It is mainly due to the mass employment of depthwise separable convolution (Chollet, 2017) in which kernels are pairs of vectors and cannot be directly pruned. If we replace all the depthwise separable convolutions with normal convolutions, the micro model merely gains accuracy advantage of 0.3% over the macro model. However, it instead contains 67.8M parameters, which is more than 4 times of macro (16.3M). Also, it will consume more than 28GB of memory space to perform the layer-hierarchy search. LDS results (Figure 2) show that add-on neurons at lower layers work better, which indicates that rich representation of low-level features is crucial to the performance of NHN. When comparing the final test accuracy (96.52%) to the network without any add-on neuron (96.20%), we know that add-on neurons are helpful in increasing the performance of NHN. In fact, perturbation on the add-on genes discovered by LDS almost always leads to degradation of performance, and the total ablation of added neurons in the final model causes accuracy drop of 1.08%, which proves that the search results are optimal.

The main goal of this paper is neither to comprehensively discuss the properties of neuron fields (Fischer, 1973), nor to investigate a training method on an entirely randomly generated neuron graph. We'd like to point out that it is quite possible to directly generate a large number of free neurons with somewhat arbitrary connections and train this "random neuron field" to address the same task presented in this work. However, because modern GPUs, or to be more precise, the computation softwares that run on these GPUs are mainly designed for dense 4-d tensor calculation. It is hard to efficiently train such random neuron field at present. Therefore, as sophisticated as our approach may seem, it's an efficient method to construct network architectures that highlight the significance of individual neurons and perform competitively against other state-of-the-art methods.

5 CONCLUSION

Neural networks that are designed by human experts and search algorithms perform outstandingly in image classification. However, redundancy in parameters exists widely in layer-based architectures. We propose a heuristic method to construct neuron-based architectures, namely, neuron hierarchical networks (NHN), that obviate the redundancy in weights and emphasize the relation between individual neurons. Experiments show that the NHN discovered based on ENAS and by location-direction search (LDS) outperforms the original ENAS architecture and many other handcrafted and randomly searched models in Cifar-10 classification, while requiring much fewer parameters. Also, the search time of NHN is very efficient compared to several state-of-the-art network architecture search methods, while achieving competitive performance.

REFERENCES

Karim Ahmed and Lorenzo Torresani. Connectivity learning in multi-branch networks. *ArXiv e-prints*, 2017.

Authors Anonymous. Dynamic network pruning via smooth initialization. *submitted to Neurocomputing*, 2018.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

Maura Boldrini, Camille A Fulmore, Alexandria N Tartt, Laika R Simeon, Ina Pavlova, Verica Poposka, Gorazd B Rosoklija, Aleksandar Stankov, Victoria Arango, Andrew J Dwork, et al. Human hippocampal neurogenesis persists throughout aging. *Cell stem cell*, 22(4):589–599, 2018.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. In *ICLR*, 2018.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. *ArXiv e-prints*, 2017.

Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *NIPS*, pages 4470–4478, 2017.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

Fergus IM Craik and Ellen Bialystok. Cognition through the lifespan: mechanisms of change. *Trends in cognitive sciences*, 10(3):131–138, 2006. doi: 10.1016/j.tics.2006.01.007.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *ArXiv e-prints*, 2017.

Jin-Dong Dong, An-Chieh Cheng, Da-Cheng M Juan, Wei Wei, and Min Sun. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures. In *ICLR Workshop*, 2018.

Susan Elmore. Apoptosis: a review of programmed cell death. *Toxicologic pathology*, 35(4):495–516, 2007. doi: 10.1080/01926230701320337.

Burkhart Fischer. A neuron field theory: Mathematical approaches to the problem of large numbers of interacting nerve cells. *Bulletin of Mathematical Biology*, 35(3):345–357, 1973.

Xavier Gastaldi. Shake-shake regularization of 3-branch residual networks. In *ICLR Workshop*, 2017.

David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier, 1991.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ArXiv e-prints*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *CVPR*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016b.

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *CVPR*, volume 1, page 3. IEEE, 2017.

Lukasz Kaiser, Aidan N Gomez, and Francois Chollet. Depthwise separable convolutions for neural machine translation. *ArXiv e-prints*, 2016.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009. data retrieved from CIFAR-10, `https://www.cs.toronto.edu/~kriz/cifar.html`.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *ArXiv e-prints*, 2013.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *ArXiv e-prints*, 2017.

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *ArXiv e-prints*, 2017.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. *ArXiv e-prints*, 2017.

Guo-li Ming and Hongjun Song. Adult neurogenesis in the mammalian brain: significant answers and significant questions. *Neuron*, 70(4):687–702, 2011.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ArXiv e-prints*, 2018.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *ArXiv e-prints*, 2018.

Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *NIPS*, pages 4053–4061, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ArXiv e-prints*, 2014.

Shawn F Sorrells, Mercedes F Paredes, Arantxa Cebrian-Silla, Kadellyn Sandoval, Dashi Qi, Kevin W Kelley, David James, Simone Mayer, Julia Chang, Kurtis I Auguste, et al. Human hippocampal neurogenesis drops sharply in children to undetectable levels in adults. *Nature*, 555(7696):377, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. In *CVPR*. IEEE, 2017.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *ArXiv e-prints*, 2015.

Tom Veniat and Ludovic Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. *ArXiv e-prints*, 2017.

Lingxi Xie and Alan Yuille. Genetic cnn. *ArXiv e-prints*, 2017.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 5987–5995. IEEE, 2017.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: a high-rank rnn language model. In *ICLR*, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *ArXiv e-prints*, 2016.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *ArXiv e-prints*, 2017.