arXiv:1911.09251v1 [cs.LG] 21 Nov 2019

AutoShrink: A Topology-aware NAS for Discovering Efficient Neural Architecture

Tunhou Zhang,[1] Hsin-Pai Cheng,[1] Zhenwen Li,[2] Feng Yan,[3] Chengyu Huang,[4] Hai Li,[1] Yiran Chen[1]
[1]ECE Department, Duke University, Durham, NC 27708
[2]Institute of Computational Linguistics, Peking University, Beijing, China
[3]CSE Department, University of Nevada, Reno, NV 89557
[4]Department of Electronic Engineering, Tsinghua University, Beijing
{tunhou.zhang,dave.cheng,hai.li,yiran.chen}@duke.edu,
lizhenwen@pku.edu.cn, fyan@unr.edu, huangcy16@mails.tsinghua.edu.cn

Abstract

Resource is an important constraint when deploying Deep Neural Networks (DNNs) on mobile and edge devices. Existing works commonly adopt the cell-based search approach, which limits the flexibility of network patterns in learned cell structures. Moreover, due to the topology-agnostic nature of existing works, including both cell-based and node-based approaches, the search process is time consuming and the performance of found architecture may be sub-optimal. To address these problems, we propose *AutoShrink*, a topology-aware Neural Architecture Search (NAS) for searching efficient building blocks of neural architectures. Our method is node-based and thus can learn flexible network patterns in cell structures within a topological search space. Directed Acyclic Graphs (DAGs) are used to abstract DNN architectures and progressively optimize the cell structure through edge shrinking. As the search space intrinsically reduces as the edges are progressively shrunk, *AutoShrink* explores more flexible search space with even less search time. We evaluate *AutoShrink* on image classification and language tasks by crafting *ShrinkCNN* and *ShrinkRNN* models. ShrinkCNN is able to achieve up to 48% parameter reduction and save 34% Multiply-Accumulates (MACs) on ImageNet-1K with comparable accuracy of state-of-the-art (SOTA) models. Specifically, both ShrinkCNN and ShrinkRNN are crafted within 1.5 GPU hours, which is 7.2× and 6.7× faster than the crafting time of SOTA CNN and RNN models, respectively.

1 Introduction

Neural Architecture Search (NAS) emerged only in recent years but has already demonstrated great strength in designing neural architectures automatically. Many research works show that neural architectures obtained from NAS surpass the performance of the hand-crafted counterpart for challenging tasks, such as computer vision and natural language processing (Liu, Simonyan, and Yang 2018; Long et al. 2015; Tan et al. 2019; Tan and Le 2019). The study on NAS methods in early stage concentrated on seeking large-scale neural architectures that can provide record breaking performance (Long et al. 2015). As computational power and computing time are not taken into consideration, the redundancy is inevitable for the models obtained by these methods.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

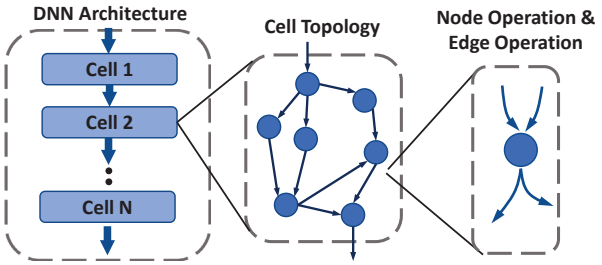


Figure 1: Neural architecture structure and cell topology. A DNN architecture is composed of several cell structures (Cell 1, Cell 2, ...). The *cell topology* of a cell denotes the network connectivity patterns within the cell structure. For a *cell topology*, operators are abstracted as nodes and tensor distributions are taken as edges, through which the tensors are communicated.

Figure 1 depicts the neural architecture hierarchical structure and its building block – cell structures, the topology of which can be descried as node operations and the connectivity between nodes, i.e., edge operations. When searching for more efficient neural architectures, the cell-based search approach is commonly employed (Tan and Le 2019; Cai, Zhu, and Han 2018). These methods adopt the architecture motifs from hand-crafted models (e.g., MobileNetV2) as backbone structures. The found architectures have compact structure and compelling performance. However, the cell-based approach heavily relies on existing cell structures and has constrained search space. It is not able to further discover the topology of an existing cell structure, which is likely to induce performance degradation.

While the node-based approach does not depend on existing cell structures, the topology-agnostic mechanism, i.e., pre-defined node graph topology, is utilized to optimize the corresponding operations (Liu, Simonyan, and Yang 2018; Pham et al. 2018). It is impossible to fully explore the network connectivity patterns within cell structures. The recently proposed randomly wired neural networks (Xie et al. 2019) utilizes a random graph priors to facilitating the interaction of tensors in DNN architectures. However, its topology-agnostic nature makes the crafted models prone to structural redundancy and sub-optimal performance.

In this paper, we propose *AutoShrink*—a topology-aware NAS methodology. By exploring the cell topology within cell structures, we aim to improve the performance and efficiency of found neural architecture and avoid the search space explosion due to the increased cell topology dimension. More specific, *AutoShrink* adopts a node-based search strategy by abstracting DNN operations as nodes in Directed Acyclic Graphs (DAGs) and the distribution of tensors as edges between nodes. The search starts with a complete DAG and leverage its interconnected topology to fully utilize the flow of tensor between nodes. To reduce the risk of space explosion, we introduce a topology knowledge accumulation mechanism and progressively optimize the cell structure through edge shrinking. *AutoShrink* can explore the significantly larger and more flexible search space with less search time as the search space is intrinsically reduced with the shrinking of edges.

AutoShrink supports a wide range of applications, including Convolutional Neural Network (CNN)-based and Recurrent Neural Network (RNN)-based models. We prototype *AutoShrink* and conduct a case study for crafting mobile-friendly neural architectures, where efficiency plays a critical role due to the highly constraint computing resources. We evaluate *AutoShrink* for CNN and RNN architecture search on image classification and language tasks, respectively. ShrinkCNN that is crafted over the ImageNet-1K dataset (Deng et al. 2009) has the similar accuracy performance as state-of-the-art (SOTA) techniques. Meanwhile, it has only $3.6\mathrm{M}$ parameters, 48% or 25% reduction compared to the hand-crafted MobileNetV2 ($6.9\mathrm{M}$) (Sandler et al. 2018) or MNasNet-A ($4.8\mathrm{M}$) (Tan et al. 2019), respectively. In terms of computational cost, ShrinkCNN cuts of 34% MACs compared to MobileNetV2 while providing the similar accuracy. ShrinkRNN, a RNN model discovered using the Penn-Treebank dataset (Marcus et al. 1994), achieves competitive performance with SOTA models by taking only 1.5 GPU hours search time, which is $6.7\times$ faster than ENAS (Pham et al. 2018) and $16\times$ faster than the first-order DARTS (Liu, Simonyan, and Yang 2018).

2 Related Work

Neural Architecture Search (NAS) promotes the design of SOTA and efficient neural architectures by exploring combinations of node operations, activation functions, etc. Some existing works (Cai, Zhu, and Han 2018; Wu et al. 2019; Tan et al. 2019) adopt the cell-based neural architecture search which reuses architecture motifs from hand-crafted architectures. Despite such architecture motifs help reduce the search space, the fixed topology of existing cell structures severely restrict the utilization of information flow between nodes, which may lead to degraded performance of found architectures.

Recent studies (Pham et al. 2018; Liu, Simonyan, and Yang 2018) explore the node-based approach, which relaxes the design space to a combination of node operations instead of constructing with predefined cells. However, all these explorations take a topology-agnostic mechanism: the node graph topology is fixed before kicking off the search process and optimizing the corresponding operations. As a result, structural and topological knowledge cannot be explored and accumulated during the search process, leading to sub-optimal performance and long search time.

Wired Neural Architectures is proposed recently and attracted a lot of attention (Xie et al. 2019; Wortsman, Farhadi, and Rastegari 2019; Cheng et al. 2019). Random wiring (Xie et al. 2019) shows that random graphs generated by stochastic network generators can provide strong priors for network connectivity patterns. The flow of tensors in such a random wiring allows more feature interaction and thus enhances the performance of found DNN architectures. However, unconstrained random wiring may lead to an explosion of memory consumption due to the aggregation of tensors (e.g., addition and concatenation) in DNN operations. More importantly, network generators can only learn the hyperparameters used to generate the random graphs (e.g., edge connection probability, degree sequence distribution of each node, etc.), rather than the actual knowledge of the graph topology.

Network Morphism attempts to morph the architecture of a neural network but keeps its functionality (Wei et al. 2016; Gordon et al. 2018). These methods mainly focus on morphing the depth, width, and kernel size of a network, or optimizing parallel towers in Inception-like DNN architectures (Szegedy et al. 2016). The knowledge of cell topology is hardly inherited. Moreover, structure optimization based on network morphism can be time-consuming as it takes thousands of optimization steps to explore a competitive child network.

3 Methodology

3.1 An Overview of *AutoShrink* Workflow

Our proposed *AutoShrink* is a topology-aware NAS methodology for discovering efficient DNN architecture. More specifically, it explores the cell topology by progressively removing those redundant edges that do not contribute much to the model performance. Such an approach indeed present a knowledge accumulation mechanism for the topology. As such, *AutoShrink* is able to search larger and more flexible space by paying reasonable search time.

Figure 2 depicts the workflow of *AutoShrink* for a cell topology that is initially abstracted as a complete DAG. We progressively optimize its structure through the following four phases iteratively.

- Phase A: For a cell topology, construct the *topology shrink space* by removing one edge at a time and collecting all the derived candidates.
- Phase B: Randomly select K candidate cell structures from the *topology shrink space* and formulate K according to DNN architectures.
- Phase C: Collect the search metrics on the proxy dataset and use it to identify the best candidate cell topology from the K randomly chosen candidates.
- Phase D: Update the cell topology for the next iteration with the best one identified in Phase C. Because the new cell topology is derived by removing one edge from the previous iteration, the update process is called as *edge shrinking*.

The *AutoShrink* workflow is an iterative process and eventually stops when no edge exists in the cell topology of in-

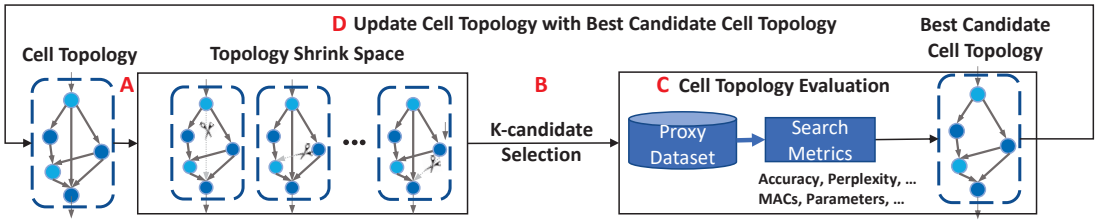
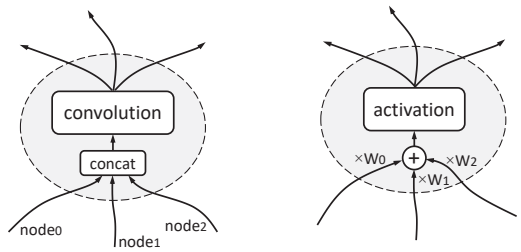


Figure 2: An overview of *AutoShrink* workflow. **A:** Construct a *topology shrink space* for a *cell topology* by iteratively removing one edge. **B:** Randomly select K-candidate from the *topology shrink space*. **C:** These selected candidates are evaluated by efficiency-aware search metric. **D:** The architecture that performs the best in evaluation is used to update the cell topology. The different node colors in a cell topology represent different operations.



(a) CNN node operation (b) RNN node operation

Figure 3: Node operations for CNN and RNN. Input nodes are first aggregated and processed by the DNN operation. The output will then be distributed to other connected nodes.

terest. More details of the workflow is elaborated in Sections 3.2∼3.4. The best candidate cell topologies obtained at the end of all the iterations will be collected and used for the post-shrink architecture construction, see Section 3.5.

3.2 Cell Topology

The main optimization objective of the proposed *AutoShrink* is cell topology, which represents the network connectivity patterns of a cell structure. A cell topology is denoted as a DAG $G = (\mathcal{V}, \mathcal{E})$, where DNN operations are abstracted as *nodes* ($\mathcal{V}$) and the distribution of tensors are represented as *edges* ($\mathcal{E}$) connecting nodes.

A node operation denotes a DNN operation o_v that processes the aggregation of tensors from other nodes and produces an output tensor x_v. o_v is parameterized by its weight parameters w_v. It can be a convolutional layer when searching for CNN architectures, or a recurrent layer followed by a unique activation function, such as ReLU and tanh, when searching for RNN architectures. Unlike random wiring (Xie et al. 2019) that assigns identical DNN operation for each node v, we assign each node $v \in \mathcal{V}$ a unique DNN operation o_v to expand the search space.

Figure 3(a) illustrates a CNN node operation. Existing works (Xie et al. 2019) apply element-wise addition during the aggregation of tensors from different nodes. To preserve the interaction of tensors, we use filter concatenation to aggregate tensors from different nodes and enable the flow of information between different layers. The aggregated tensor is then processed by a convolution operation to produce an output tensor for the current node. Given k input tensors $x_1, x_2, ..., x_k$, the node v computes the output tensors as:

$$x_v = o_v(Concat[x_1, x_2, ..., x_k]), \tag{1}$$

where $Concat$ denotes the filter concatenation.

Figure 3(b) gives an example of RNN node operation. Each of the input tensors is passed through a recurrent layer for a transformation. The transformed tensors are then summed to form an aggregated tensor, which is passed through a non-linear activation function to produce the output tensor for this node. Inspired by the memory mechanism in Recurrent Highway Networks (RHN) (Zilly et al. 2017), we use a highway bypass between adjacent nodes to get memory state. Given k input tensors $x_1, x_2, ..., x_k$, the node v computes the output tensors as:

$$c_i = \sigma(x_i \cdot w_i^1), \tag{2}$$

$$x_v = \sum_{i=1}^{k} c_i \otimes a(x_i \cdot w_i^2) + (1 - c_i) \otimes x_i, \tag{3}$$

where $\otimes$ denotes the element-wise multiplication, w_i^1 is highway gate parameters, and w_i^2 is transform matrices. σ denotes the $sigmoid$ activation function, and a represents the assigned activation function for the current node.

A DAG that represents a cell topology can be directly mapped to an unique DNN building block. During the mapping, a node that does not have any input connections are dropped. The output for the building block can be constructed from the leaf nodes with zero out-degree. For CNN architectures, the leaf nodes are concatenated within the last dimension to produce an output feature map for the building block. For RNN architectures, the leaf nodes are averaged to produce an output feature map.

3.3 Topological Search Space Construction

Random wiring (Xie et al. 2019) uses identical DNN operation (e.g., a 3×3 separable convolution) for all the nodes. Our proposed *AutoShrink* removes this constraint and constructs a topological search space to enable the exploration of more flexible cell topology.

In CNN architecture search, each node can choose either 1×1 convolution or depth-wise separable 3×3 convolution

as its operation. As batch normalization (Ioffe and Szegedy 2015) speed up DNN training, every convolution operation adopts a Convolution-BatchNorm-ReLU triplet. For RNN architectures, we extend the search space by randomly assigning unique non-linear function to each node following the recurrent layer. The non-linear functions include ReLU, sigmoid, tanh, and identity mapping.

3.4 Edge Shrinking

AutoShrink progressively optimizes the cell structure in a crafted topology shrink space. In every iteration, *AutoShrink* compares the performance of candidate structures and accumulates topological knowledge.

Topology shrink space is used to describe the possible topological reduction during edge shrinking for improving the cell structure. The topology shrink space π is defined as a full set of all possible cell structures that can be derived by applying graph damage (i.e., removing one edge from the existing graph) to the current cell topology $g^{(t)} = (V^{(t)}, E^{(t)})$ at time t, such as

$$\pi(g^{(t)}) = \{(V^{(t)}, E^{(t)} - \{e\}) | \forall e \in E^{(t)}\}. \quad (4)$$

Candidate cell structures. Considering the large topology shrink space induced by the node connection possibilities at time t, we adopt K-candidate selection strategy. It randomly picks only K candidate cell structures from the topology shrink space, and accumulates the topological knowledge from only the best candidate cell structure according to the search metrics. This aggressive optimization immensely reduces the topology shrink space. As our results in Section 4.3 shall show, such an aggressive reduction in topology shrink space does not degrade much performance because the accumulated topological knowledge could compensate for the missing in the architecture evaluations. Despite of the reduction of topology shrink space, the intrinsic reduction on the overall search space is a leading factor to the lower search cost. With the combination of the above two factors, the search cost of *AutoShrink* is significantly reduced.

Shrink process. For a cell topology g, the shrink process targets to optimize it within the topology shrink space based on a resource-aware search metric:

$$S(g) = Perf(\mathcal{A}(g); \mathcal{D}) - \lambda \cdot \log Res(\mathcal{A}(g)), \quad (5)$$

where $\mathcal{A}(g)$ denotes the neural architecture built with g, and $\mathcal{D}$ is the proxy dataset used for evaluation. $Perf()$ represents the best validation performance we can achieve on the proxy dataset. For CNN architectures for image classification tasks, *accuracy* can be taken as the performance metric. For RNN architectures targeting on language tasks, the performance metric can be *perplexity*. $Res()$ denotes the resource consumption of a model, such as the number of parameters or MACs of $A(g)$. λ is an adjustable parameter which penalizes the resource consumption to form a lightweight neural architecture.

To search for efficiency-aware neural architectures, *AutoShrink* incorporates the resource-aware metric as a continuous penalty function into the search metric. By evaluating the performance of candidate architectures in the topology shrink space and picking the best one according to the search metric, the current cell structure can better utilize the topological knowledge from similar structures and make improvement by adapting to the best candidate cell structure in the topology shrink space. The pseudo code for the shrink process is given in Algorithm 1.

Algorithm 1 *AutoShrink*

Input:
N: Number of nodes in the initial complete DAG.
K: Number of candidates in topology shrink space to evaluate in each shrink step.
begin
  Generate a complete graph $g^{(0)}$ with N nodes and randomly assign DNN operations.
  $t \leftarrow 0$
  while $E^{(t)} \neq \{\emptyset\}$ **do**
    Phase A: Construct the topology shrink space $\pi(g^{(t)})$
    Phase B: Adopt K-candidate selection strategy to select K candidate cell structures $g_1^{(t)}, g_2^{(t)}, ..., g_K^{(t)}$.
    Phase C: Construct the candidate DNNs using all candidate cell structures $g_1^{(t)}, g_2^{(t)}, ..., g_K^{(t)}$.
    Train the candidate DNNs on proxy dataset and get the feedback search metrics $S(g_1^{(t)}), S(g_2^{(t)}), ..., S(g_K^{(t)})$..
    Phase D: Use the best candidate cell structure to update cell topology.
    $g^{(t+1)} \leftarrow \arg\max_{g' \in \{g_1^{(t)}, g_2^{(t)}, ..., g_K^{(t)}\}} S(g')$.
  end while
  return g
end

Topology knowledge accumulation. The progressive improvement process accumulates the topological knowledge in the cell structure as the edge shrinking steps forward. The accumulation of topology knowledge is demonstrated in two aspects. On the one hand, redundant edges are sequentially moved out of the cell topology to facilitate the exploration of efficient but representative cell topology. On the other hand, due to the reduction of total number of edges in the cell topology, the search space is becoming smaller for the cell topology to make further improvement. For example, the initial search space is estimated to contain 6.8×10^{10} neural architectures for a given complete DAG with 28 edges, 8 nodes and 2 possible choices for each node operation as the cell topology. After one shrink step, at most 27 edges still exist in the cell topology and the search space is reduced by at least $4\times$, which contains at most 1.7×10^{10} neural architectures. Smaller search space enables a faster exploration of representative cell structure, which can be otherwise unexplored within limited time budget.

More formally, at time t, the progressive improvement of the current cell structure g^t within the topology shrink space $\pi(g^{(t)})$ can be expressed as:

$$g^{(t+1)} = \arg\max_{g' \in \pi(g^{(t)})} S(g'). \quad (6)$$

3.5 Crafting DNN Architecture

Based on the performance metric S in Eq. (5), the cell structure with the best performance g^{opt} will be taken as our representative cell structure of optimal performance:

$$g^{opt} = \arg\max_{\hat{g} \in \mathbf{g}} S(\hat{g}). \tag{7}$$

This representative cell structure is used as the building block to construct DNN architectures.

Modularization for CNN. As CNNs are well-known to be a hierarchical design with different feature map size in different stages, we divide a DNN architecture into multiple stages. Instead of strided convolution, MaxPooling is used as the down-sampling module to connect two adjacent stages. Table 1 gives an example of modularizing *AutoShrink* cells to construct CNN architectures for the CIFAR-10 task (Krizhevsky and others 2009). In each stage, we stacked T representative cells found by *AutoShrink* (i.e., g^{opt}). The width of these cells remains the same within a stage, and doubles when passing the down-sample module. Furthermore, a residual connection (He et al. 2016) is added from the input node to the output node within our optimal CNN cell structure, which is helpful to develop extra network connectivity patterns. With the incorporation of residual connections, the performance grows with the depth of DNN architecture like ResNets (He et al. 2016).

Modularization for RNN. To ensure the maximum flexibility in RNN architecture construction. we do not assume any repetitive patterns while constructing optimal RNN architectures from optimal RNN cell structures. The final RNN architecture for Penn Treebank consists of an embedding layer, an optimal RNN cell explored by *AutoShrink*, and a decoder to generate the final predictions. Following DARTS (Liu, Simonyan, and Yang 2018), we use a fully connected layer to construct the decoder so that the hidden state can be used to predict.

4 Experiments and Discussion

We implement and evaluate *AutoShrink* for CNN and RNN architecture search respectively on image classification and language tasks. Considering there are $7 \sim 12$ nodes in SOTA node-based NAS (Pham et al. 2018; Liu, Simonyan, and Yang 2018), we set the initial cell structure to be a complete DAG with $N = 8$ nodes for CNN search and $N = 6$ for RNN search. Empirically, we set λ to 0.1 to favor efficiency-aware architecture in both CNN and RNN search.

Hierarchy	Output resolution	Regime
Stem CONV	32×32	CONV 3×3 32 filters
MP + Stage 1	32×32	*AutoShrink* Cell, T, 16 filters
MP + Stage 2	16×16	*AutoShrink* Cell, T, 32 filters
MP + Stage 3	8×8	*AutoShrink* Cell, T, 64 filters
Classifier	1×1	AP, FC, Softmax

Table 1: The ShrinkCNN architecture configuration for the CIFAR-10 task. CONV denotes the Convolution-BatchNorm-ReLU triplet. MP and AP denotes MaxPooling and Average Pooling respectively. On CIFAR-10 task, ShrinkCNN has 3 stages with T *AutoShrink* cells stacked in each stage.

4.1 *ShrinkCNN* for Image Classification

Representative CNN cell structures from proxy dataset. We construct the proxy dataset for image classification tasks by randomly selecting 5,000 examples from the CIFAR-10 dataset (Krizhevsky and others 2009) with an equal distribution of classes. To obtain representative cell structures, we first build candidate CNN architectures based on the candidate cell structures obtained from the *AutoShrink* process. A candidate architecture follows the configuration in Table 1: it has three stages; each stage contains one *AutoShrink* cell ($T = 1$); and the numbers of convolutional filters in the three stages are 16, 32, and 64, respectively.

These candidate neural architectures are trained on the proxy dataset. MAC is taken as the focused optimization resource and integrated into the search metrics. In each shrink step, we adopt a K-candidate selection strategy with K set to 10 to deliver a fast architecture search without sacrificing the performance of cell topology. Each training takes about 20 epochs to reach convergence. We then evaluate the validation accuracy on the proxy dataset to get the feedback search metrics. The largest candidate architecture derived from our proxy dataset has a computational cost of $20\mathrm{M}$ MACs and takes about 40 GPU seconds to reach convergence.

Once the *AutoShrink* process is completed, the cell structure that provides the best result of search metric is selected as the representative cell structure. As shown in Figure 4, the representative CNN cell structure consists of 4 node operations and 3 filter concatenations. The left part of the structure develops a Siamese pair of convolutions with wired concatenations to produce similar feature maps for the following different-sized convolutions. The right part utilizes the previous input, and learns a combination of different-sized filters to capture the spatial information.

ShrinkCNN on ImageNet-1K. The representative CNN cell structure is adapted to ImageNet-1K dataset (Deng et al. 2009) for crafting the optimal network architecture, namely, *ShrinkCNN*. In this work, we craft two architectures. ShrinkCNN-A is obtained by using ReLU activation, which can provide a fair comparison of *AutoShrink* and the ReLU-based CNN architectures (Sandler et al. 2018; Liu, Simonyan, and Yang 2018; Xie et al. 2019). ShrinkCNN-

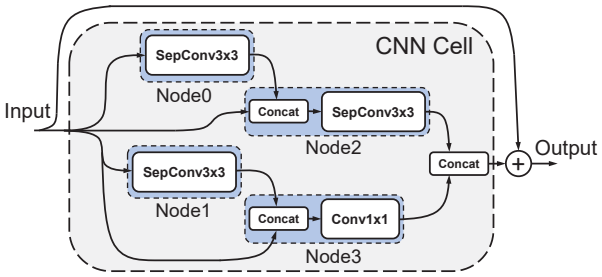


Figure 4: The representative CNN cell structure.

Architecture	Top-1 Error	# Param (M)	MACs (G)
ResNet-50	**24.0%**	26	4.1
MorphNet	24.8%	15.5	-
MobileNetV1	29.4%	4.2	0.569
MobileNetV2	25.3%	6.9	0.585
MnasNet-A	24.4%	4.8	**0.340**
EfficientNet-B0	23.7%	5.3	0.391
DARTS	26.7%	4.7	0.574
RandWire-WS	25.3%	5.6	0.583
ShrinkCNN-A	26.1%	**3.6**	0.385
ShrinkCNN-B	24.9%	**3.6**	0.385

Table 2: Performance comparison of various CNN architectures on ImageNet-1K dataset. All the evaluations are based on 50,000 images of ImageNet-1K validation dataset. The input resolution is set to 224×224.

Architecture	Perplexity		#Param	Search Cost
	valid	test	(M)	(GPU hours)
LSTM	60.7	58.8	24	-
LSTM-SC	60.9	58.3	24	-
RHN	67.9	65.4	23	-
ENAS	-	55.8	23	10
DARTS	58.1	55.7	23	24
ShrinkRNN	58.5	56.5	23	1.5

Table 3: Performance comparison of various neural architectures on Penn Treebank dataset. We evaluate ShrinkRNN on both the validation and test dataset for fair comparison.

B is crafted by using swish (Ramachandran, Zoph, and Le 2017) activation that helps increase the representation power of models. We compare ShrinkCNN-B with the swish-based CNN architectures (Tan and Le 2019; Tan et al. 2019).

We adopt similar pre-processing pipeline as Inception-V3 (Szegedy et al. 2016) and use RMSprop optimizer (Hinton, Srivastava, and Swersky 2012) with an initial learning rate 0.1 to optimize the CNN architectures. The cosine learning decay suggested in SGDR (Loshchilov and Hutter 2016) is employed to reduce the generalization error.

Performance evaluation. We compare ShrinkCNN-A and ShrinkCNN-B with SOTA hand-crafted and automatically searched models. Table 2 summarizes their key performance metrics on the ImageNet-1K dataset. In general, ShrinkCNN requires fewer parameters and MAC operations while providing the similar accuracy. For example, compared to the hand-crafted MobileNetV2 model with ReLU activation, ShrinkCNN-A reduces 48% model parameters and 34% MACs. Compared to MNasNet-A which is crafted by conducting resource-aware neural architecture search, ShrinkCNN-B can further cut off 25% parameters with negligible impact on the top-1 error rate. Our ShrinkCNN-B requires slightly more MACs mainly because MnasNet-A applies architecture motifs (e.g., squeeze-and-excitation layers) to further increase the efficiency of MACs. Such architecture motifs can also be combined with ShrinkCNN-B to further boost the performance. Here we only show the performance results of *AutoShrink* without prior knowledge to demonstrate its effectiveness. Compared to EfficientNet (Tan and Le 2019) obtained by scaling up MobileNetV2 blocks using automated search, ShrinkCNN-B can save 32% parameters with the similar top-1 error rate and MACs.

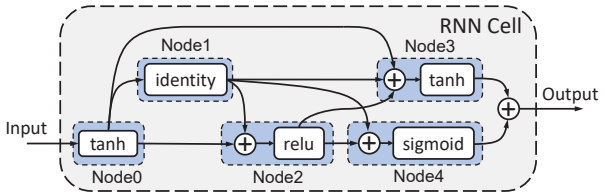


Figure 5: The representative RNN cell structure.

4.2 *ShrinkRNN* for Language Tasks

Representative RNN cell structures from proxy dataset. In this work, the proxy dataset for language tasks is constructed by randomly selecting 4,000 sentences from the Penn Treebank dataset. We adopt the similar approach for CNN architectures in the search of representative RNN cell structure. Specifically, for language tasks, we set the dimension of embedding and hidden units of candidate cell structures to 200 (Liu, Simonyan, and Yang 2018; Pham et al. 2018) and incorporate the number of parameters as efficiency-aware resource consumption into the search metric. In each shrink step, we adopt a K-candidate selection strategy with K set to 5 to balance the search speed and the performance of cell topology. The training of candidate neural architecture on the proxy dataset takes about 10 epochs to converge. The largest RNN architecture derived from our candidate cell structures has 1.2 Million parameters and takes about 80 GPU seconds to reach convergence.

Figure 5 depicts the found representative RNN cell structure. It has 5 node operations and 4 node additions. The left node combines the hidden state from the previous step and the input word embedding. Each node on the right side learns a combination of others' outputs to capture high-level information. This RNN cell integrates various activation functions, which improves its representational power.

ShrinkRNN on Penn Treebank. We adapt our representative RNN cell structure to the full Penn-Treebank dataset (Marcus et al. 1994) for crafting *ShrinkRNN*. For comparison purpose, we rescale ShrinkRNN to a fixed setting of 23 Million parameters. NT-ASGD algorithm (Merity, Keskar, and Socher 2018) is used to train the ShrinkRNN architecture and the initial learning rate is set to 20. Additional regularization techniques include an ℓ_2 regularization weighted by 8×10^{-7}; variational dropout (Gal and Ghahramani 2016) of 0.2 to word embeddings, 0.75 to cell input, 0.25 to hidden nodes and 0.75 to output layer.

Performance evaluation. We compare ShrinkRNN with SOTA models including LSTM (Merity, Keskar, and Socher 2018), LSTM with skip connections (LSTM-SC) (Melis, Dyer, and Blunsom 2018), recurrent highway network (RHN) (Zilly et al. 2017), ENAS (Pham et al. 2018), and

DARTS (Liu, Simonyan, and Yang 2018). Table 3 summarizes the validation and test results on the Penn-Treebank dataset. ShrinkRNN provides a comparable performance while its crafting time is significantly shorter (over $6.7\times$ shorter) than the existing node-based NAS methods.

4.3 Ablation Studies

Efficiency of CNN cell topology. We analyze the progressive optimization procedure towards our representative CNN cell structure to evaluate the efficiency of our cell topology. Starting with a complete DAG with $N = 8$ nodes and 28 edges, *AutoShrink* progressively optimizes the CNN cell topology to only 6 edges. The computational cost of our representative cell structure reduces from 6.12M MACs to 2.60M MACs. To verify the effectiveness of our knowledge accumulation mechanism, we compare the performance of neural architectures constructed from complete-DAG-based cell topology and strong topology priors with our representative cell topology. The topology priors we use are Watts-Strogatz (WS) graph, Erdös-Rényi (ER) graph, and Barabási-Albert (BA) graph, which are popular in graph theory.

We craft CNN architectures with the above topology respectively, train them on the full CIFAR-10 dataset with the same hyperparameter setting, and compare their test results with ShrinkCNN in Table 4. Table 4 shows that ShrinkCNN is able to achieve up to 1.66% higher accuracy than the complete-DAG-based topology, and up to 3% higher than the topology from random graphs, thanks to the topological knowledge accumulation in the *AutoShrink* process.

Efficiency of RNN cell topology. We also analyze the progressive optimization procedure in RNN cell structure. Starting with complete DAG with 6 nodes and 15 edges, *AutoShrink* progressively optimize the cell topology to only 8 edges, while the number of parameters within the cell topology is reduced from 3.37M to 2.8M. We craft RNN architectures using complete-DAG-based topology and our representative cell structures. Then, we train the RNN architectures on full Penn Treebank dataset with the same hyperparameters. Table 5 indicates that our representative RNN cell structure can achieve a significantly lower (6.6) perplexity with 30% fewer parameters.

K-candidate selection in CNN. We further investigate the impact of K value in the K-candidate selection strategy and analyze its impact on both search cost and performance. Specifically, we configure K to be 5, 10, 15 while running *AutoShrink* on complete DAGs with 6, 8, 10 nodes. From figure 6, we observe that different combinations of the number of nodes and K candidates may result in slightly difference in final accuracy, although increasing K can strengthen the candidate selection process. However, the search cost dramatically increases with K. For example, when $N = 8$, changing K from 10 to 15 increases the searching time from 1.5 hours to 1.875 hours, while the validation accuracy has subtle difference.

Topology	Nodes	Accuracy (%)	MAC (M)
Complete-DAG	8	91.56	105.61
WS	15	90.5±0.41	16.04±2.71
ER	15	92.1±0.38	34.40±16.43
BA	15	90.0±0.34	32.61±5.96
ShrinkCNN	8	93.22	38.20

Table 4: Comparison with ShrinkCNN topology with prior graph topology. We use $N = 8$ nodes when crafting CNN architectures with complete-DAG-based topology, and $N = 15$ nodes when crafting CNN architectures with random graph priors. Experiments on random graph priors are conducted 10 times to reduce randomness as much as possible.

Cell Topology	Edges	Perplexity	# Param (M)
Complete-DAG	15	63.1	33
ShrinkRNN	8	56.5	23

Table 5: Demonstration of efficient RNN cell structures found by *AutoShrink* algorithm on full Penn Treebank dataset. We used $N = 6$ nodes when crafting RNN architectures with complete-DAG-based topology.

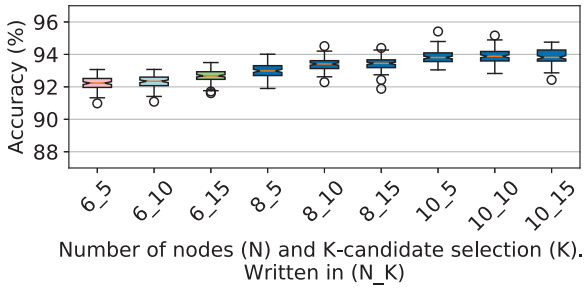


Figure 6: The impact of K on the performance of our representative CNN structures with different number of nodes at the beginning of *AutoShrink* process. Marginal performance gain is spotted when increasing K to conduct more delicate *AutoShrink* process with larger search cost.

5 Conclusion

In this work, we propose a topology-aware NAS method, *AutoShrink*, for discovering efficient and representative cell topology to formulate DNN architectures. To explore significantly larger and more flexible space, *AutoShrink* accumulates topology knowledge by combining intrinsic search space reduction and K-candidate selection strategy to significantly reduce the search cost. The above contributions make *AutoShrink* consistently outperform both hand-crafted models and automatically searched models on both image classification and language tasks without any prior knowledge. Using representative cells found by *AutoShrink*, the crafted ShrinkCNN achieves comparable accuracy on ImageNet-1K dataset with up to 48% parameter reduction and up to 34% MACs reduction compared to SOTA models. The crafted ShrinkRNN has comparable performance results as SOTA models while can be crafted $6.7\times$ faster.

Acknowledgements. This project is in part supported by the following grants: NSF-1937435, NSF CCF-1725456, DOE SC0017030, Qualcomm Gift, NSF CCF-1756013, IIS-1838024 (using resources provided by Amazon Web Services as part of the NSF BIGDATA program). We also thank USTC for the support of computing.

References

[Cai, Zhu, and Han 2018] Cai, H.; Zhu, L.; and Han, S. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

[Cheng et al. 2019] Cheng, H.-P.; Zhang, T.; Yang, Y.; Yan, F.; Teague, H.; Chen, Y.; and Li, H. 2019. Msnet: Structural wired neural architecture search for internet of things. *International Conference on Computer Vision Workshop on Neural Architects*.

[Deng et al. 2009] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[Gal and Ghahramani 2016] Gal, Y., and Ghahramani, Z. 2016. A theoretically grounded application of dropout in recurrent neural networks. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc. 1019–1027.

[Gordon et al. 2018] Gordon, A.; Eban, E.; Nachum, O.; Chen, B.; Wu, H.; Yang, T.-J.; and Choi, E. 2018. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1586–1595.

[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

[Hinton, Srivastava, and Swersky 2012] Hinton, G.; Srivastava, N.; and Swersky, K. 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

[Ioffe and Szegedy 2015] Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[Krizhevsky and others 2009] Krizhevsky, A., et al. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

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

[Long et al. 2015] Long, M.; Cao, Y.; Wang, J.; and Jordan, M. I. 2015. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*.

[Loshchilov and Hutter 2016] Loshchilov, I., and Hutter, F. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

[Marcus et al. 1994] Marcus, M.; Kim, G.; Marcinkiewicz, M. A.; MacIntyre, R.; Bies, A.; Ferguson, M.; Katz, K.; and Schasberger, B. 1994. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, 114–119. Association for Computational Linguistics.

[Melis, Dyer, and Blunsom 2018] Melis, G.; Dyer, C.; and Blunsom, P. 2018. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*.

[Merity, Keskar, and Socher 2018] Merity, S.; Keskar, N. S.; and Socher, R. 2018. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*.

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

[Ramachandran, Zoph, and Le 2017] Ramachandran, P.; Zoph, B.; and Le, Q. V. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

[Sandler et al. 2018] Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

[Szegedy et al. 2016] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

[Tan and Le 2019] Tan, M., and Le, Q. V. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

[Tan et al. 2019] Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2820–2828.

[Wei et al. 2016] Wei, T.; Wang, C.; Rui, Y.; and Chen, C. W. 2016. Network morphism. In *International Conference on Machine Learning*, 564–572.

[Wortsman, Farhadi, and Rastegari 2019] Wortsman, M.; Farhadi, A.; and Rastegari, M. 2019. Discovering neural wirings. *arXiv preprint arXiv:1906.00586*.

[Wu et al. 2019] Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; and Keutzer, K. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 10734–10742.

[Xie et al. 2019] Xie, S.; Kirillov, A.; Girshick, R.; and He, K. 2019. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*.

[Zilly et al. 2017] Zilly, J. G.; Srivastava, R. K.; Koutník, J.; and Schmidhuber, J. 2017. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 4189–4198. JMLR. org.