Topology-Aware Network Pruning using Multi-stage Graph Embedding and Reinforcement Learning

Sixing Yu [1] **Arya Mazaheri** [2] **Ali Jannesari** [1]

Abstract

Model compression is an essential technique for deploying deep neural networks (DNNs) on power and memory-constrained resources. However, existing model-compression methods often rely on human expertise and focus on parameters' local importance, ignoring the rich topology information within DNNs. In this paper, we propose a novel multi-stage graph embedding technique based on graph neural networks (GNNs) to identify DNN topologies and use reinforcement learning (RL) to find a suitable compression policy. We performed resource-constrained (i.e., FLOPs) channel pruning and compared our approach with state-of-the-art model compression methods. We evaluated our method on various models from typical to mobile-friendly networks, such as ResNet family, VGG-16, MobileNet-v1/v2, and ShuffleNet. Results show that our method can achieve higher compression ratios with a minimal fine-tuning cost yet yields outstanding and competitive performance. The code is open-sourced at https://github.com/yusx-swapp/GNN-RL-Model-Compression.

1. Introduction

The demand for deploying DNN models on edge devices (e.g., mobile phones, robots, and self-driving cars) is expanding rapidly. However, the increasing memory and computing power requirements of DNNs make their deployment on edge devices a grand challenge. Thus, various custom-made DNN models have been introduced by experts to accommodate a DNN model with reasonably high accuracy on mobile devices (Howard et al., 2019; Tan & Le, 2019; Zhang et al., 2018b; Ma et al., 2018; Mehta et al., 2020; Huang et al., 2018). In addition to mobile-friendly deep networks, model compression methods such as network pruning, have been considerably useful by introducing sparsity or eliminating channels or filters. Nevertheless, it requires extensive knowledge and effort to find the perfect balance between accuracy and model size.

The main challenge of network pruning is to find the best pruning schedule or strategy. Furthermore, a pruning strategy for a given DNN is not transferable to a different DNN, demanding a customized per-network pruning strategy. Recently, various pruning methods (He et al., 2018b; Yu et al., 2021) have been proposed to automatically compress DNNs. However, they either use manually defined rules/embeddings, ignoring rich topological information, or do not consider topology changes while model compression. Moreover, since RL-based methods (Liu et al., 2020; He et al., 2018b; Yu et al., 2021) usually use the pruned model accuracy as RL agent's reward function, a negative correlation emerges between the compression ratio and reward. Consequently, without any constraint, the RL agent tends to search for a tiny compression ratio to get a better reward. To address this problem and get the desired model size reduction, existing RL-based methods often need additional heuristic algorithms to adjust the pruning ratio.

The computational representation of DNNs often contains various patterns (a.k.a. motifs) repeated throughout the network topology. For instance, MobileNetV2 involves 17 blocks, each following a similar graph and operation structure. The topology of such blocks can represent their states, allowing us to exploit their redundancy level and importance. Such structural information inspired us to model a given DNN as hierarchical computational graphs and propose multi-stage graph neural networks (m-GNN) for DNN embedding. Additionally, we equipped m-GNN with a reinforcement learning agent (GNN-RL) to automatically search for the compression policy (i.e., pruning ratios). To avoid tiny compression ratios due to the negative correlation between the compression ratio and the RL agent's reward, we created a DNN-Graph environment for the GNN-RL agent. Such an environment allows the agent to continuously com-

[1]Department of Computer Science, Iowa State University, Iowa, US [2]Department of Computer Science, Technical University of Darmstadt, Darmstadt, Germany. Correspondence to: Sixing Yu <yusx@iastate.edu>, Arya Mazaheri <arya.mazaheri@tu-darmstadt.de>, Ali Jannesari <jannesar@iastate.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

press the DNNs until it satisfies the model size constraint. For each step of the compression, the DNN-Graph environment converts the compressed DNN to a graph. The graph is the environment state input to the GNN-RL agent. Once the compressed DNN satisfies the desired model size, the DNN-Graph ends the search episodes and uses the pruned DNN's accuracy as a reward for the agent. We successfully performed FLOPs-constraint network pruning on various DNNs and achieved competitive results with the state-of-the-arts pruning methods. More importantly, the experiments showed that the learned topology of a given DNN could be transferred to another DNN. Such a feature proves that graph embedding is applicable to network pruning.

In essence, this paper makes the following contributions:

- A novel method for modeling DNNs as hierarchical graphs to exploit their topological and structural information for topology-aware network pruning.
- An efficient multi-stage GNN (m-GNN) to learn hierarchical and transferable graph embeddings.
- A simpler yet efficient RL method based on the PPO algorithm for adaptive network pruning.
- Competitive results with the state-of-the-art model compression methods on various DNN models.

2. Related Work

Various studies focus on model compression and efficient deployment of DNNs, such as network pruning (Han et al., 2016; He et al., 2018b; Li et al., 2020a; Chin et al., 2020; Guo et al., 2020; Ye et al., 2020; Zhuang et al., 2020; Tang et al., 2020; Ning et al., 2020a; Chen et al., 2021; Lai et al., 2021; Gao et al., 2021; Liu et al., 2021; Wang et al., 2021; Chen et al., 2020), knowledge distillation (Hinton et al., 2015), and network quantization (Gholami et al., 2021). Within the scope of this paper, we mainly focus on structured network pruning (Anwar et al., 2017), as it is not bound to special AI accelerators (Zhang et al., 2018a; Guo et al., 2016). Uniform, shallow, deep empirical structured pruning policies (He et al., 2017; Li et al., 2016), the handcrafted structured pruning methods, such as SPP (Wang et al., 2017), FP (Li et al., 2016), and RNP (Lin et al., 2017) fall into the structured pruning category. However, such pruning policies often fail to work properly on new models and might lead to sub-optimal performance. Recently, AutoML pruning algorithms (Li et al., 2020a; Chin et al., 2020; Ye et al., 2020; Chen et al., 2020; Li et al., 2020c; Chin et al., 2020; Lin et al., 2020; Li et al., 2020b) offered better results with higher versatility, particularly the RL-based methods (Liu et al., 2020; He et al., 2018b; Yu et al., 2021). Liu et al. (Liu et al., 2020) proposed an ADMM-based (Boyd et al., 2011) structured weight pruning method and an innovative additional purification step for further weight reduction. He et al. (He et al., 2018b) proposed AMC and used RL to predict each hidden layer's compression policy. However, they manually defined DNN embeddings, such as the number of input/output channels, parameter size, and FLOPs for the RL environment state vectors, and ignored the neural network's essential structural information. Yu et al. (Yu et al., 2021) modeled DNNs as graphs and introduced a GNN-based graph encoder-decoder to embed DNNs' hidden layers. Nevertheless, they performed layer-wise pruning based on simple layer embeddings, ignoring the global topology changes when pruning. Moreover, they construct simplified computational graphs for DNNs and do not take advantage of motifs in DNNs.

Graph Neural Networks (GNN). GNN and its variants (Kipf & Welling, 2017; Schlichtkrull et al., 2018) can learn the graph embeddings and have been successfully used for link prediction (Liben-Nowell & Kleinberg, 2007) and node classification. However, these methods are mainly focused on node embedding and are inherently flat, which is inefficient to deal with the hierarchical data. In this paper, we aim to learn the global topology information from DNNs. Thus, we proposed multi-stage GNN (m-GNN), which takes advantage of the repetitive motifs available in DNNs. m-GNN considers the edge features and has a novel learning-based pooling strategy to learn the global graph embedding.

Graph-based Neural Architecture Search (NAS). Although this paper is not directly related to NAS, it is an active area of research wherein the computationally expensive operations are replaced with a more efficient alternative (Li et al., 2021a;b; Yao et al., 2021; Wang et al., 2020). Particularly, graph-based NAS methods apply GNN and use graph-based neural architecture encoding schemes to exploit neural network's topology. They model neural architecture's search spaces as graphs and aim to search for the best performing neural network structure (Guo et al., 2019; Shi et al., 2019; Dudziak et al., 2021; Chatzianastasis et al., 2021; Ning et al., 2020b). Such methods inspired us to exploit compression policy from the topology information of DNNs.

3. Approach

To prune a given DNN, the user provides the model size constraint (i.e., FLOPs constraint). Although we perform FLOPs-constraint filter pruning, our method is not limited to FLOPs-constraint and can be easily extended to latency, MACs, or sparsity constraint compression.

Figure 1 illustrates the DNN-Graph search episode, which is essentially a model compression iteration. Red arrows show that the process starts from the original DNN. The

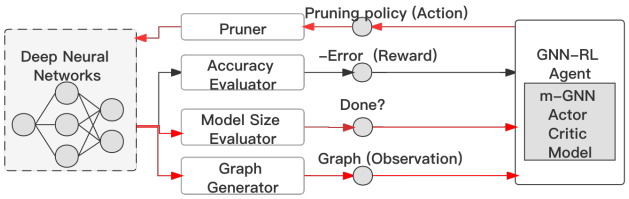

Figure 1. An overview of DNN-Graph environment search episode.

model size evaluator first evaluates the size of the DNN. If the size is not satisfied, the graph generator converts the DNN into a hierarchical computational graph. Then, the GNN-RL agent leverages m-GNN to learn pruning ratios from the graph. The pruner prunes the DNN with the pruning ratios and begins the next iteration from the compressed DNN. Each step of the compression will change the network topology. Thus, the DNN-Graph environment reconstructs a new hierarchical computational graph for the GNN-RL agent corresponding to the current compression state. Once the compressed DNN satisfies the size constraint, the evaluator will end the episode, and the accuracy evaluator will assess the pruned DNN's accuracy as an episode reward for the GNN-RL agent. As opposed to the existing RL-based methods (He et al., 2018b; Yu et al., 2021; Liu et al., 2020), with the DNN-Graph environment, the GNN-RL can learn to reach the desired model size in the reinforcement learning stage. Hence, it prevents us from adjusting pruning ratios and obtaining tiny compression ratios. In the following, we will explain the details of the m-GNN and RL agent within our approach.

3.1. Hierarchical Graph Representation

Computational graphs with their rich topological information may involve billions of operations (He et al., 2016), making them bloated and hard to understand. Nevertheless, such graphs often contain repetitive sub-graphs (a.k.a. motifs), such as 3×3 convolutions or custom blocks in the state-of-the-art networks. We aim to simplify computational graphs by extracting the motifs and modeling them as hierarchical computational graphs. Additionally, we coarsen the graph by replacing primitive operations such as *add*, *multiple*, and *minus* with machine-learning high-level operations (e.g., convolution, pooling).

We formally model a given DNN as an l-level hierarchical computational graph, such that at the l^{th} level (the top level), we would have the hierarchical computational graph set $\mathcal{G}^l = \{G^l\}$, where each item is a computational graph $G^l = (V^l, \mathcal{E}^l, \mathcal{G}^{l-1})$. V^l is the graph nodes corresponding to hidden states. $\mathcal{E}^l$ is the set of directed edges with a specific edge type associated with the operations. Lastly, $\mathcal{G}^{l-1} = \{G_0^{l-1}, G_1^{l-1}, ...\}$ is the computational graph set at the $(l-1)$-level as well as the operation set at layer l. The hierarchical computation graph's size depends on the primitive operations we choose in $\mathcal{G}^0$. In this paper, we opted for commonly used machine-learning operations as the primitive operations for $\mathcal{G}^0$. As an example, Figure 2 illustrates the idea behind generating hierarchical computational graphs using a sample graph G, where the edges are operations and the nodes are hidden states. In the input graph, we choose three primitive operations $\mathcal{G}^0 = \{1\times1$ conv, 3×3 conv, 3×3 max-pooling$\}$ corresponding to the three edge types. Then, we extract the repetitive subgraphs (i.e., G_1^1, G_2^1 and G_3^1), each denoting a compound operation, and decompose the graph G into two hierarchical levels, as shown in Figure 2 (b) and (c).

In practice, a 2-layer hierarchy is suitable for representing a DNN when our target is a convolutional layer. In the first layer, we represent the convolution operation and construct motifs, and in the second layer, we use the motifs to construct the DNN we aim to prune.

3.2. Multi-stage GNN

Standard GNN and its variants (Kipf & Welling, 2017) are inherently flat (Ying et al., 2018). Since we model a given DNN as an $l-$level hierarchical computational graph, we propose a multi-stage GNN (m-GNN), which embeds the hierarchical graph in l-stages according to its hierarchical levels and analyzes the motifs. As depicted in Figure 2, m-GNN initially learns the lower level embeddings and uses them as the corresponding edge features in high-level computation graphs. Instead of learning node embeddings, m-GNN aims to learn the global graph representation. We further introduced a novel learning-based pooling strategy for every stage of embedding. With m-GNN, we only need embedding once for each motif on the computational graph. It is much more efficient and uses less memory than embedding a flat computation graph with standard GNN.

Multi-stage embedding. For the computational graphs $\mathcal{G}^t = \{G_0^t, G_1^t, ..., G_{N_t}^t\}$ in the t^{th} hierarchical level, we embed the computational graph $G_i^t = (V_i^t, \mathcal{E}_i^t, \mathcal{G}^{t-1}), i = \{1, 2, ..., N_t\}$ as:

$$e_i^t = EncoderGNN_t(G_i^t, E_{t-1}), \tag{1}$$

where e_i^t is the embedding vector of G_i^t, $E_{t-1} = \{e_j^{t-1}\}, j = \{1, 2, ..., N_{t-1}\}$ is the embedding of the computational graphs at level $t-1$. We use E_{t-1} as edge features at level t. For level-1, E_0 contains the initial features (e.g., one-hot, and random standard) of the primitive operations $\mathcal{G}^0$ that we manually select. In the hierarchical computational graphs, each edge corresponds to a computational graph of the previous level and uses its graph embedding as the edge feature. Furthermore, the graphs at the same hierarchical level share the GNN's parameter. At the top level (l^{th} level) of the hierarchical graph $\mathcal{G}^l = \{G^l\}$, we only have one computational graph and its embedding is the

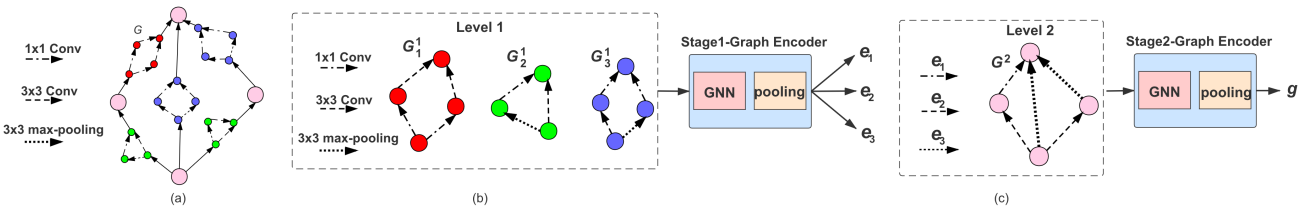

Figure 2. A two-level hierarchical computational graph and m-GNN. The sub-graphs are painted with red, blue, and green colors.

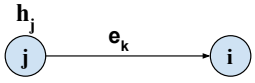

Figure 3. Message passing from node j to node i.

DNN's final embedding g:

$$g = EncoderGNN_l(G^l, E_{l-1}) \tag{2}$$

Message passing. In the multi-stage hierarchical embedding, we consider the edge features. However, in the standard graph convolutional networks (GCN) (Kipf & Welling, 2017), it only passes the node features and the message passing function can be formulated as follows:

$$h_i^{l+1} = \sum_{j \in N_i} \frac{1}{c_i} W^l h_j^l, \tag{3}$$

where h is nodes' hidden states, c_i is a constant coefficient, N_i is node i neighbors, and W^l is GNN's learnable weight matrix. Instead of standard message passing, in the multi-stage GNN, we add the edge features:

$$h_i^{l+1} = \sum_{j \in N_i} \frac{1}{c_i} W^l (h_j^l \circ e_k^{l-1}), \tag{4}$$

where e_k^{l-1} is the features of edge (i, j) and is also the embeddings of the k^{th} graph at level $l-1$, such that the edge (i, j) corresponds to the operation G_k^{l-1}. The operation $\circ$ denotes the element-wise product.

Many message-passing strategies exist, such as MP-GNN (Gilmer et al., 2017), in which they utilized a multi-layer perceptron. In this work, assuming the scenario depicted in Figure 3, the message passing between h_j and e_k should essentially capture the amount of information in the node j (i.e., the node feature h_j) that can flow to the node i by using the factor e_k. Thus, we selected element-wise product as the message passing function to assure that the same edge type can flow the same amount of information. Additionally, element-wise product satisfies the associative property in multi-stage message passing.

Learning-based pooling. A typical GNN aims to learn the node embeddings of a graph (e.g., learning node representation and perform node classification). However, our goal is to learn the graph representation of a given DNN. Thus, we introduced a learning-based pooling method for multi-stage GNN to learn the graph embedding from node embeddings. We define the graph embedding e as:

$$e = \sum_{i \in N} \alpha_i h_i + \sum_{j \in D} \alpha_j \vec{0}, \tag{5}$$

where N is the set of nodes, h_i is the i^{th} node embedding, D is the set of pruned nodes, and α_i is the learnable weight coefficient. In the multi-stage GNN, the computational graphs at the same hierarchical level share the GNN's parameters, but in the pooling, each computational graph has its own learnable pooling parameters α. This parameter contains shared weights before and after pruning, and its dimension will not change. However, the number of nodes is not fixed after pruning, causing matrix dimension mismatch between the learning parameter α and the node features h (i.e, $|\alpha| \neq |h|$). As a remedy, we keep the size of h constant by replacing the pruned node's features with a zero vector.

Training. In GNN-RL, m-GNN is part of the actor-critic network inside the RL's policy network and will be updated end-to-end using the PPO algorithm (see section 3.3 for detail).

3.3. Network Pruning Using Reinforcement Learning and m-GNN

We employed m-GNN together with reinforcement learning (RL) to find suitable network pruning strategies. In the following, we explain the details of the RL agent.

Environment states. We use the generated hierarchical computational graph $\mathcal{G}^l$ for representing the DNN's state and the RL agent's environment state. Since pruning the model causes its underlying graph topology to change, the DNN-Graph environment constantly updates the graph $\mathcal{G}^l$ after each pruning step to help the RL agent find the pruning policy on the current state.

Action space. The actions made by the RL agent are pruning ratios within a continuous space. Specifically, the GNN-RL agent's action space $A \in \mathbb{R}^{N \times 1}$, where N is the number of pruning layers, is the pruning ratios for hid-

den layers: $A = [a_1, a_2, \ldots, a_N]^T$, where $a_i \in [0, 1)$ and $i = \{1, 2, ..., N\}$ is the pruning ratio for i^{th} layer. GNN-RL agent makes the actions directly from the topology states:

$$g = GraphEncoder(\mathcal{G}^l), \tag{6}$$

$$A = MLP(g), \tag{7}$$

where $\mathcal{G}^l$ is the environment states, g is the graph representation, and MLP is a multi-layer perceptron neural network. The graph encoder learns the topology embedding, and the MLP projects the embedding into hidden layers' pruning ratios. To bound the network's output within the action space, we apply a clamping (a.k.a. clipping) function within the range of [0,1) to the actions.

Reward function. The reward function is $R_{err} = -Error$, where $Error$ is the compressed DNN's Top-1 error on the validation set. In computing the reward, we do not consider the model size, as the graph environment will automatically stop the search episode when the RL agent reaches the desired size.

RL policy. Various RL policies aim to search within a continuous action space, such as proximal policy optimization (PPO) (Schulman et al., 2017) and deep deterministic policy gradient (DDPG) (Lillicrap et al., 2016). Although various state-of-the-art methods use the DDPG RL policy to search for the best pruning policy, we opted for the PPO RL policy, as it provided much better performance. m-GNN is part of the actor-critic network inside the GNN-RL agent and will be updated end-to-end using the PPO algorithm. Equation 8 shows the objective function that we used in the PPO update policy. m-GNN is then trained by minimizing the objective function using the gradient descent algorithm.

$$L(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{8}$$

where θ is the policy parameter of the RL agent, which includes the m-GNN's parameters, $\hat{\mathbb{E}}_t$ denotes the empirical expectation over time steps, $r_t(\theta)$ is the ratio of the probability under the new and old policies, $\hat{A}_t$ is the estimated advantage at time t, and ϵ is a clip hyperparameter, usually set to 0.1 or 0.2.

4. Experimental Results

To show the effectiveness of the GNN-RL, we evaluate our approach on various deep networks and compare our method with the following methods:

- Traditional channel reduction methods, such as uniform empirical policies, SPP (Wang et al., 2017), FP (Li et al., 2016), RNP (Lin et al., 2017), FPGM (He et al., 2019), SFP (He et al., 2018a), DSA (Ning et al., 2020a) and PFP (Liebenwein et al., 2020).
- AutoML methods, such as NetAdapt (Yang et al., 2018), AutoPruner (Jian-Hao Luo, 2020), EagleEye (Li et al., 2020a), AutoSlim (Yu & Huang, 2019), MetaPruning (Liu et al., 2019), AMC (He et al., 2018b), AGMC (Yu et al., 2021), and random search (RS) with RL.

4.1. Implementation Details

Graph representation settings. We model a given DNN as 2-layer hierarchical graphs, the primitive operations $\mathcal{G}^0 = \{k \times k$ conv, depth-wise conv, point-wise conv, $k \times k$ max-pooling$\}$. When pruning, instead of using a one-hot vector, we initialized the hierarchical computational graph's node and edge features using the standard distribution with a feature size of 20.

RL agent settings. We use the Adam optimizer to update the RL agent's actor-critic network, where the learning rate is 3×10^{-4} and the $\beta = (0.9, 0.999)$. Moreover, we update the policy every 100 search episodes, and in each updating round, we train the RL agent for 20 epochs. The discount factor is $\gamma = 0.99$, the clip parameter is 0.2, and the standard deviation of actions is 0.5. Actor and critic networks contain a graph encoder with (hidden, embedding) size of (50,50) units and a multi-layer perceptron with a hidden size of 200 units. Although the output dimension of the actor-network is the number of actions, the critic-network's MLP has only one output dimension.

Dataset settings. The experiment involves multiple datasets, including CIFAR-10/100 (Krizhevsky & Hinton, 2009), and ImageNet (Russakovsky et al., 2015). In the CIFAR-10/100 dataset, we sample $5K$ images from the test set as the validation set. In ImageNet (ILSVRC-2012), we split $10K$ images from the test set as the validation set. When searching, the DNN-Graph environment uses the compressed model's R_{err} on the validation set as the GNN-RL agent's reward.

Network settings. Since ResNet contains residual connections between convolutional layers, different pruning ratio among residual connected layers leads to feature maps mismatch. Instead of removing the residual connections, we share the pruning ratio between residual connected layers (i.e., equal pruning ratio in residual connected layers). For MobileNet networks, applying regular filter pruning on depth-wise/point-wise convolution layers causes information loss. Thus, instead of pruning such filters separately, we only prune linear expansion layers and point-wise filters within MobileNet blocks. Since residual connections are between linear expansion layers in MobileNet-v2, we share the linear expansion layers' pruning ratio. Lastly, to prune ShuffleNet networks, we consider its blocks together and perform channel pruning inside the blocks. In a ShuffleNet block, we do not prune the expansion layer (the output layer of the block), which can preserve the number of output

Table 1. Top-1 accuracy results for pruned ResNets on CIFAR-10 and ShuffleNet on CIFAR-100. ΔTop-1 shows the top-1 accuracy gap between the pruned and the original model.

Model	Method	FLOPs ↓	Top-1	ΔTop-1
Res110	AGMC	50%	93.08	−0.60
	RS	50%	87.26	−6.42
	PFEC	39%	93.30	−0.20
	SFP	41%	93.38	+0.16
	FPGM	52%	93.74	−0.60
	GNN-RL	52%	94.31	+0.63
Res56	Uniform	50%	87.5	−5.89
	AMC	50%	90.20	−3.19
	FPGM	59%	93.26	−0.02
	AGMC	50%	92.00	−1.39
	EagleEye	50%	94.66	N/A
	GNN-RL	54%	93.49	+0.10
Res32	AGMC	50%	90.96	−1.67
	RS	50%	89.57	−3.06
	SFP	42%	92.08	−0.55
	FPGM	50%	91.93	−0.70
	GNN-RL	51%	92.58	−0.05
Res20	Uniform	50%	84.00	−7.73
	AMC	50%	86.40	−5.33
	AGMC	50%	88.42	−3.31
	SFP	42%	90.83	−1.37
	FPGM	42%	91.09	−1.11
	DSA	50%	91.38	−0.79
	GNN-RL	51%	91.31	-0.42
Shuff-v1	AGMC	40%	65.26	−3.38
	RS	40%	63.70	−4.94
	GNN-RL	42%	67.10	−2.84
Shuff-v2	AGMC	40%	66.28	−2.57
	RS	40%	65.74	−3.11
	GNN-RL	46%	66.64	−2.21

channels.

Fine-tuning settings. As we observed a direct correlation between the pre-/post-fine-tuning accuracy, when pruning on CIFAR-10/100, we perform fine-tuning after pruning. However, the validation accuracy on the ImageNet dataset is sensitive to the compression ratio, particularly for the MobileNet-v1/2. Without fine-tuning, high compression ratios lead to a considerable accuracy drop. Thus, on MobileNet-v1/2 trained on ImageNet, we perform one additional fine-tuning epoch before we compute the reward to ensure that the RL agent gets a valuable reward (as depicted in Figure 5, one epoch of fine-tuning can recover the majority of accuracy loss). After pruning, a 150-epochs fine-tuning process is applied to the pruned DNNs on ImageNet/CIFAR-100, and a 100-epochs fine-tuning is applied on CIFAR-10. We use the SGD optimizer, where the $\alpha = 5 \times 10^{-3}$, $B = 512$, and the weight decay is 5×10^{-4}. In each epoch, the cosine learning rate decay is applied.

Table 2. Top-1 accuracy results for pruned models on ImageNet. ΔTop-1 shows the top-1 accuracy gap

Model	Method	FLOPs ↓	Top-1	Δ Top-1
VGG16	FP	80%	55.90	−14.6
	RNP	80%	66.92	−3.58
	SPP	80%	68.20	−2.30
	AMC	80%	69.10	−1.40
	AutoPruner	74%	69.20	−2.39
	GNN-RL	80%	70.99	+0.49
Res18	FPGM	42%	68.41	-1.87
	SFP	42%	67.10	-3.18
	PFP-B	43%	65.65	-4.09
	GNN-RL	51%	68.66	-1.10
Res50	AutoPruner	44%	73.84	-2.31
	Meta-Pruning	50%	73.40	-3.20
	AutoSlim	50%	74.00	-2.10
	EagleEye	50%	74.20	N/A
	GNN-RL	53%	74.28	-1.82
MB-v1	Uniform	25%	68.40	−2.20
	NetAdapt	34%	69.10	−1.50
	AMC	34%	70.50	−0.40
	Meta-Pruning	35%	70.60	0
	EagleEye	25%	70.90	N/A
	GNN-RL	30%	70.70	−0.20
	AMC	60%	68.90	−2.00
	GNN-RL	60%	69.50	−1.40
MB-v2	AMC	27%	70.80	−1.00
	Meta-Pruning	27%	71.20	−3.50
	NetAdapt	25%	70.00	N/A
	GNN-RL	42%	70.04	−1.83

4.2. Comparisons with State-of-the-art

Tables 1 and 2 compare the pruning efficiency of GNN-RL with state-of-the-art methods trained on CIFAR-10/100 and ImageNet datasets. Results show that GNN-RL achieves competitive results compared with state-of-the-art methods and even outperforms many of them either by higher accuracy or pruning ratio. Pruning ResNet-110/56 models even caused higher test accuracy than the original model, which could be due to over-fitting, as the accuracy on the training set was 100%. To verify our assumption, we explored the relationship between the FLOPs constraints and the accuracy. Figure 4 shows that the 66%-FLOPs Resnet-110 can get the highest test accuracy. When the FLOPs reduction ratio exceeds 0.66, the test accuracy drops intensively.

We further analyzed the redundancy and the importance of ResNet layers. Figure 6 shows the hidden layers' pruning ratios on ResNet-110 and ResNet-56. GNN-RL agent automatically learns that the residual connection layers with ResNet are redundant. Thus, it applies more pruning on such layers. Moreover, GNN-RL's results are inconsistent with previous handcrafted pruning works, which assume that the deeper layers are more important for final predictions and tend to prune less. However, GNN-RL applies more prun-

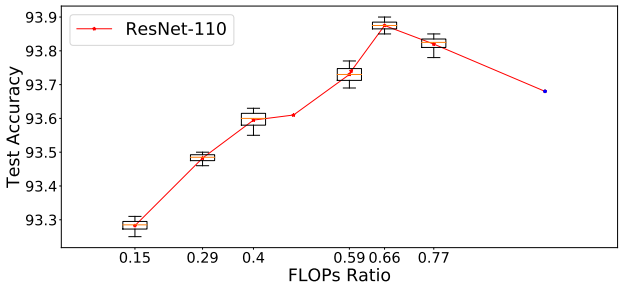

Figure 4. Test accuracy of ResNet-110 using various FLOPs ratios.

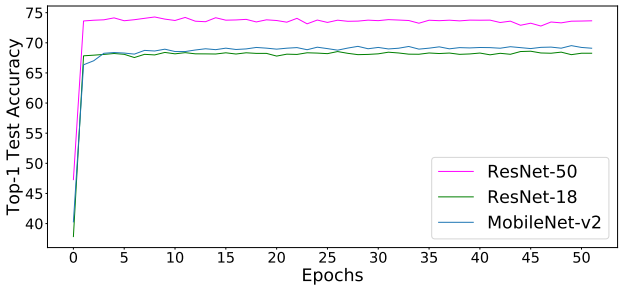

Figure 5. Accuracy recovery after different fine-tuning epochs.

ing to the middle (layers 45 to 65) and deep (layers 90 to 109) layers within ResNet-110. Such an observation proves that handcrafted rules are not generalizable to all DNNs, particularly those with residual connections.

4.3. Ablation Study

Recoverability. A noteworthy feature of the models pruned by GNN-RL is that they can rapidly recover from accuracy loss. Figure 5 demonstrates the fine-tuning learning curve for ResNet-50/18 and MobileNet-v2. We noticed that after only one epoch of fine-tuning, we can recover from the accuracy loss caused by pruning. We only needed less than 50 epochs for further accuracy improvement, which is far less than the number of fine-tuning epochs used by other state-of-the-art methods.

Effectiveness of reinforcement learning policy. To the best of our knowledge, existing RL-based pruning methods (e.g., AMC) employ the DDPG policy for implementing their RL agent. However, we found out that PPO policy offers better performance and converges faster than DDPG. Figure 7 shows the RL agents' learning curve, comparing GNN-RL implemented with DDPG and PPO with AMC. In addition to better performance, PPO has less tunable hyperparameters, making PPO easier to configure and tune.

Additionally, the RL policy network of GNN-RL is a tiny neural network that contains a graph encoder and a two-layer perceptron with limited action and environment space, leading to efficient training and inference. For instance, in the experiment conducted on CIFAR-10, GNN-RL could converge within half a GPU hour using an Nvidia V100.

Table 3. ResNet-110 node classification results using GCN and m-GNN.

Graph	Nodes	Edges	Method	Acc.%
Plain	394,412	581,332	GCN	83.50
Hierarchical	12,460	20,221	m-GNN	84.20

Topology transferability. The topology transferability is a key factor to demonstrate whether GNN embeddings are necessary or even applicable. We aim to prove that GNN-RL can learn a transferable policy from a given DNN topology. Intuitively, GNNs trained on a topology can be transferred to a simpler topology. Thus, we first trained GNN-RL on ResNet-56 and then transferred the graph encoder to ResNet-44. When searching for the pruning policy, we disabled the graph encoder's gradients and only updated the MLP component, which projects the topology embeddings into the action space. Figure 8 shows the RL's learning curve for direct pruning search and the transferred policy. We noticed that the transferred GNN-RL has a similar learning curve to direct search on ResNet-44, indicating that the learned policy can be reused for networks with similar topology. Such a feature offers a rapid pruning process ($1.12\times$ faster for each round) with much less computing time, as we only need to update the MLP's parameter. Under the transfer policy, we achieved 93.23% accuracy with 51% FLOPs reduction comparable to the original ResNet-44, which is 93.10%.

The impact of hierarchical graph representation (m-GNN). We realized that motifs frequently appearing in computation graphs should have the same embedding. With m-GNN, we only need to embed each motif once. However, a plain GNN disregards this hypothesis, leading to repetitive embeddings and incurring unnecessary costs. To further prove our hypothesis, we experimented with the ResNet-110's computational graph and labeled the nodes according to their hidden layer. Table 3 shows that hierarchical representation causes to have smaller graph sizes and helps our method to achieve a higher node classification accuracy with fewer graph nodes and edges, saving a considerable amount of computing resources. Moreover, using m-GNN led to a faster convergence and as shown in Tables 1 and 2 resulted into more precise pruned models.

4.4. Inference Acceleration and Memory Saving

The inference and memory usage of compressed DNNs are essential metrics to determine the possibility of DNN deployment on a given platform. Thus, we evaluated the pruned models' inference latency using PyTorch 1.7.1 on an

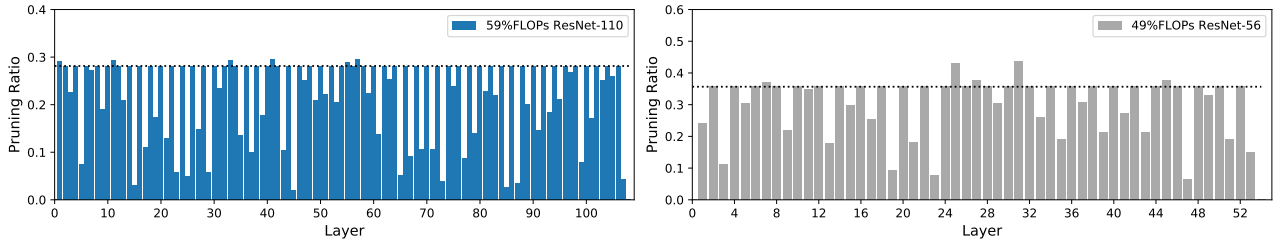

Figure 6. The hidden layers' pruning ratio of 59% FLOPs ResNet-110 and 49% FLOPs ResNet-56. The bars that tangent with the dot-line are the residual connection layers.

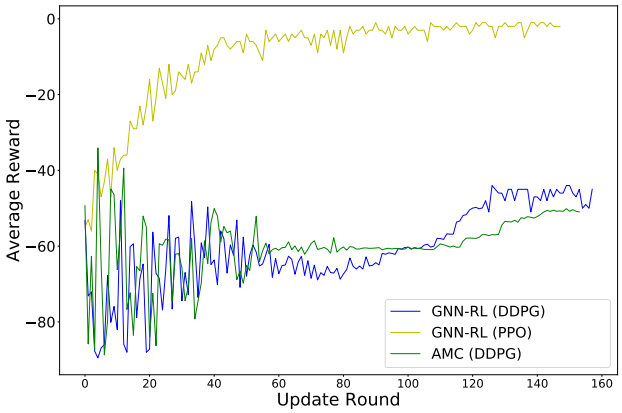

Figure 7. Learning curve of RL policies; DDPG vs. PPO.

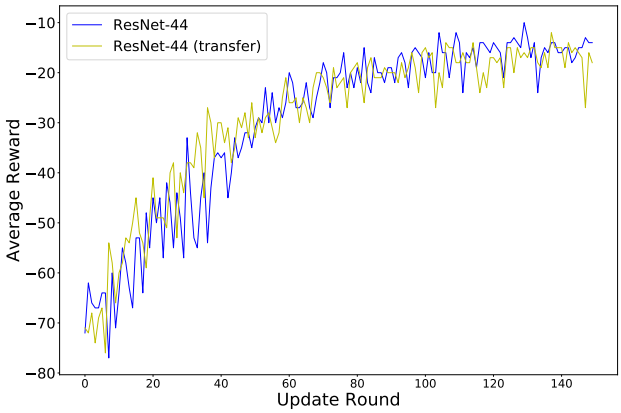

Figure 8. Topology transferability.

Nvidia GTX 1080Ti GPU and recorded the GPU memory usages. The ResNet-110/56/44/32/20, VGG-16, and MobileNet/ShuffleNet networks are evaluated on the CIFAR-10, ImageNet, and CIFAR-100 datasets with batch size 32.

Table 4 shows the inference accelerations and memory savings on our GPU. All the models pruned by GNN-RL achieve noteworthy inference acceleration and GPU memory reductions. Particularly, for the VGG-16, the original model's GPU memory usage is 528 MB since it has a very compact dense layer, which contributes little to FLOPs but leads to extensive memory requirement. The GNN-RL prunes convolutional layers and significantly reduces the feature map size, consuming 141 MB less memory than the original version. The inference acceleration on VGG-16 is also noticeable, with $1.38\times$ speed up on the ImageNet.

The inference acceleration for mobile-friendly DNNs may seem relatively insignificant. However, such models are designed for deployment on mobile devices. Thus, we believe that our tested GPU, with its extensive resources, does not take advantage of the mobile-friendly properties.

5. Conclusion

This paper proposed a neural network pruning method called GNN-RL that utilizes graph neural networks and reinforcement learning to exploit a topology-aware compression policy. We introduced the DNN-Graph environment that converts compression states to a topology modification process and allows GNN-RL to learn the desired compression ratio without human intervention. To efficiently embed DNNs and take advantage of motifs, we introduced m-GNN, a new multi-stage graph embedding method. In our experiments, GNN-RL is validated and verified on over-parameterized and mobile-friendly networks. For over-parameterized models pruned by GNN-RL, ResNet-110/56, the test accuracy even outperformed the original models, i.e. $+0.63\%$ on ResNet-110 and $+0.1\%$ on ResNet-56. For mobile-friendly DNNs, the 40% FLOPs MobileNet-v2 pruned by GNN-RL with only 1.4% test accuracy loss. Additionally, all the pruned models accelerated the inference speed and saved a considerable amount of memory usage. Most importantly, GNN-RL learns the topology transfer policy, enabling the GNN-RL to prune various DNNs with transfer learning.

Table 4. THE LATENCY AND GPU MEMORY USAGE BEFORE AND AFTER PRUNING.

MODEL	FLOPS	LATENCY	GPU MEM.
VGG-16	100%	$0.11ms$	528MB
	20%	$0.08ms$	387MB
RESNET-110	100%	$1.04ms$	6.9MB
	48%	$0.98ms$	3.4MB
RESNET-56	100%	$0.52ms$	3.4MB
	46%	$0.43ms$	1.7MB
RESNET-44	100%	$0.37ms$	2.7MB
	49%	$0.34ms$	1.3MB
RESNET-32	100%	$0.33ms$	1.9MB
	49%	$0.26ms$	942KB
RESNET-20	100%	$0.20ms$	1.1MB
	49%	$0.16ms$	548KB
MOBILENET-V1	100%	$0.22ms$	13MB
	79%	$0.19ms$	7.5MB
MOBILENET-V2	100%	$0.34ms$	9.3MB
	79%	$0.32ms$	7.4MB
SHUFFLENET-V1	100%	$0.45ms$	4.1MB
	58%	$0.43ms$	2MB
SHUFFLENET-V2	100%	$0.51ms$	5.4MB
	54%	0.50MS	2.7MB

Acknowledgement

We would like to thank the research IT team of Iowa State University for their continuous support in conducting the experiments. Experiments presented in this paper were carried out on the Pronto GPU cluster at ISU. This research has been supported by publication award of the computer science department at Iowa State University. Furthermore, we received support from Software Campus through the German Federal Ministry of Education and Research (BMBF), and the state of Hesse as part of the NHR Program.

References

Anwar, S., Hwang, K., and Sung, W. Structured pruning of deep convolutional neural networks. *Proc. of the J. Emerg. Technol. Comput. Syst.*, 13(3), February 2017. ISSN 1550-4832.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, January 2011. ISSN 1935-8237.

Chatzianastasis, M., Dasoulas, G., Siolas, G., and Vazirgiannis, M. Graph-based neural architecture search with operation embeddings. In *Proc. of the IEEE/CVF International Conference on Computer Vision (CVPR)*, pp. 393–402, 2021.

Chen, J., Chen, S., and Pan, S. J. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 14747–14758. Curran Associates, Inc., 2020.

Chen, T., Ji, B., Ding, T., Fang, B., Wang, G., Zhu, Z., Liang, L., Shi, Y., Yi, S., and Tu, X. Only train once: A one-shot neural network training and pruning framework. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 19637–19651. Curran Associates, Inc., 2021.

Chin, T.-W., Ding, R., Zhang, C., and Marculescu, D. Towards efficient model compression via learned global ranking. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Dudziak, L., Chau, T., Abdelfattah, M. S., Lee, R., Kim, H., and Lane, N. D. BRP-NAS: Prediction-based NAS using gcns, 2021.

Gao, S., Huang, F., Cai, W., and Huang, H. Network Pruning via Performance Maximization. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9270–9280, June 2021.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference, 2021.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proc. of International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Guo, S., Wang, Y., Li, Q., and Yan, J. Dmcp: Differentiable markov channel pruning for neural networks. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Proc. of the Advances in Neural Information Processing Systems*, volume 29, pp. 1379–1387. Curran Associates, Inc., 2016.

Guo, Y., Zheng, Y., Tan, M., Chen, Q., Chen, J., Zhao, P., and Huang, J. NAT: Neural architecture transformer for accurate and compact architectures. In *Proc. of the Advances in Neural Information Processing Systems*, volume 32, pp. 737–748. Curran Associates, Inc., 2019.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proc. of International Conference on Learning Representations (ICLR)*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proc. of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *Proc. of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2234–2240, 2018a.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. AMC: AutoML for model compression and acceleration on mobile devices. In *Proc. of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018b.

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *Proc. of NIPS Deep Learning and Representation Learning Workshop*, 2015.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for mobilenetv3. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019.

Huang, G., Liu, S., Van der Maaten, L., and Weinberger, K. Q. Condensenet: An efficient densenet using learned group convolutions. In *Proc. of the IEEE conference on computer vision and pattern recognition*, pp. 2752–2761, 2018.

Jian-Hao Luo, J. W. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, 2020.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2017.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

Lai, C.-I. J., Zhang, Y., Liu, A. H., Chang, S., Liao, Y.-L., Chuang, Y.-S., Qian, K., Khurana, S., Cox, D., and Glass, J. Parp: Prune, adjust and re-prune for self-supervised speech recognition. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 21256–21272. Curran Associates, Inc., 2021.

Li, B., Wu, B., Su, J., and Wang, G. Eagleeye: Fast subnet evaluation for efficient neural network pruning. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J.-M. (eds.), *Proc. of the Computer Vision – ECCV 2020*, pp. 639–654, Cham, 2020a. Springer International Publishing. ISBN 978-3-030-58536-5.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets, 2016.

Li, Y., Gu, S., Mayer, C., Gool, L. V., and Timofte, R. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8018–8027, 2020b.

Li, Y., Gu, S., Zhang, K., Van Gool, L., and Timofte, R. DHP: Differentiable meta pruning via hypernetworks, 2020c.

Li, Y., Hao, C., Li, P., Xiong, J., and Chen, D. Generic neural architecture search via regression. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 20476–20490. Curran Associates, Inc., 2021a.

Li, Z., Yuan, G., Niu, W., Zhao, P., Li, Y., Cai, Y., Shen, X., Zhan, Z., Kong, Z., Jin, Q., et al. Npas: A compiler-aware framework of unified network pruning and architecture search for beyond real-time mobile acceleration. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14255–14266, 2021b.

Liben-Nowell, D. and Kleinberg, J. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7): 1019–1031, 2007.

Liebenwein, L., Baykal, C., Lang, H., Feldman, D., and Rus, D. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *Proc. of the ICLR (Poster)*, 2016.

Lin, J., Rao, Y., Lu, J., and Zhou, J. Runtime neural pruning. In *Proc. of the Advances in Neural Information Processing Systems*, pp. 2181–2191, 2017.

Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., and Shao, L. Hrank: Filter pruning using high-rank feature map. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1529–1538, 2020.

Liu, N., Ma, X., Xu, Z., Wang, Y., Tang, J., and Ye, J. AutoCompress: An automatic dnn structured pruning framework for ultra-high compression rates. In *Proc. of the Artificial Intelligence Conference (AAAI)*, pp. 4876–4883, 2020.

Liu, Y., Liu, L., Lin, C., Dong, Z., and Wang, W. Learnable Motion Coherence for Correspondence Pruning. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3237–3246, June 2021.

Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.-T., and Sun, J. Metapruning: Meta learning for automatic neural network channel pruning. In *Proc. of the IEEE International Conference on Computer Vision*, pp. 3296–3305, 2019.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proc. of the European conference on computer vision (ECCV)*, pp. 116–131, 2018.

Mehta, S., Hajishirzi, H., and Rastegari, M. Dicenet: Dimension-wise convolutions for efficient networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Ning, X., Zhao, T., Li, W., Lei, P., Wang, Y., and Yang, H. DSA: More efficient budgeted pruning via differentiable sparsity allocation. In *Proc. of 16th European Computer Vision Conference*, pp. 592–607. Springer, 2020a.

Ning, X., Zheng, Y., Zhao, T., Wang, Y., and Yang, H. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *Proc. of European Conference on Computer Vision*, pp. 189–204. Springer, 2020b.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In Gangemi, A., Navigli, R., Vidal, M.-E., Hitzler, P., Troncy, R., Hollink, L., Tordai, A., and Alam, M. (eds.), *The Semantic Web*, pp. 593–607, Cham, 2018. Springer International Publishing. ISBN 978-3-319-93417-4.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Shi, H., Pi, R., Xu, H., Li, Z., Kwok, J. T., and Zhang, T. Bridging the gap between sample-based and one-shot neural architecture search with BONAS, 2019.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proc. of the International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Tang, Y., Wang, Y., Xu, Y., Tao, D., XU, C., Xu, C., and Xu, C. Scop: Scientific control for reliable neural network pruning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 10936–10947. Curran Associates, Inc., 2020.

Wang, H., Zhang, Q., Wang, Y., and Hu, R. Structured probabilistic pruning for deep convolutional neural network acceleration. *British Machine Vision Conference*, 2017.

Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., and Han, S. Apq: Joint search for network architecture, pruning and quantization policy. In *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2075–2084, 2020.

Wang, Z., Li, C., and Wang, X. Convolutional neural network pruning with structural redundancy reduction. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14913–14922, June 2021.

Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. Netadapt: Platform-aware neural network adaptation for mobile applications. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Yao, L., Pi, R., Xu, H., Zhang, W., Li, Z., and Zhang, T. Joint-detnas: Upgrade your detector with nas, pruning and dynamic distillation. In *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10170–10179, 2021.

Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., and Liu, Q. Good subnetworks provably exist: Pruning via greedy forward selection. In III, H. D. and Singh, A. (eds.), *Proc. of the 37th International Conference on Machine Learning*, volume 119 of *Proc. of Machine Learning Research*, pp. 10820–10830. PMLR, 13–18 Jul 2020.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Proc. of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 4805–4815, Red Hook, NY, USA, 2018. Curran Associates Inc.

Yu, J. and Huang, T. Autoslim: Towards one-shot architecture search for channel numbers, 2019.

Yu, S., Mazaheri, A., and Jannesari, A. Auto graph encoder-decoder for neural network pruning. In *Proc. of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6362–6372, October 2021.

Zhang, T., Ye, S., Zhang, K., Tang, J., Wen, W., Fardad, M., and Wang, Y. A systematic DNN weight pruning framework using alternating direction method of multipliers. *ECCV*, 2018a.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proc. of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018b.

Zhuang, T., Zhang, Z., Huang, Y., Zeng, X., Shuang, K., and Li, X. Neuron-level structured pruning using polarization regularizer. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9865–9877. Curran Associates, Inc., 2020.