Crafting Efficient Neural Graph of Large Entropy

Minjing Dong[1] , **Hanting Chen**[2,3] , **Yunhe Wang**[3] , **Chang Xu**[1]
[1]School of Computer Science, Faculty of Engineering, University of Sydney, Australia
[2]Key Laboratory of Machine Perception (Ministry of Education), Peking University, China
[3]Huawei Noah's Ark Lab
mdon0736@uni.sydney.edu.au, htchen@pku.edu.cn, yunhe.wang@huawei.com, c.xu@sydney.edu.au

Abstract

Network pruning is widely applied to deep CNN models due to their heavy computation costs and achieves high performance by keeping important weights while removing the redundancy. Pruning redundant weights directly may hurt global information flow, which suggests that an efficient sparse network should take graph properties into account. Thus, instead of paying more attention to preserving important weight, we focus on the pruned architecture itself. We propose to use graph entropy as the measurement, which shows useful properties to craft high-quality neural graphs and enables us to propose efficient algorithm to construct them as the initial network architecture. Our algorithm can be easily implemented and deployed to different popular CNN models and achieve better trade-offs.

1 Introduction

The success of Convolutional Neural Networks (CNNs) comes with massive parameters computation and storage. A wide variety of models with deeper architecture have been exploited in recent years and have achieved state-of-the-art performance in many computer vision applications, such as image classification and object detection. [Krizhevsky *et al.*, 2012; Szegedy *et al.*, 2014; Simonyan and Zisserman, 2014; He *et al.*, 2016; Huang *et al.*, 2016] However, due to the high computation costs and run-time memory, those deep networks cannot be directly deployed to some resource constrained platforms, such as mobile devices and embedded sensors, which has great application potential.

Thus, reducing the storage and computation usage of deep CNN models has received increasing attention [LeCun *et al.*, 1990; Hassibi and Stork, 1993]. Recently, some compression algorithms have been further explored to achieve satisfactory performance in deeper and large-scale CNN model compression [Han *et al.*, 2015; Li *et al.*, 2016; Zhou *et al.*, 2016; Yang *et al.*, 2016; Luo *et al.*, 2017; Liu *et al.*, 2017; You *et al.*, 2017; He *et al.*, 2017; Yu *et al.*, 2017; Zhang *et al.*, 2018; Wu *et al.*, 2018; Bansal *et al.*, 2018]. By pruning the neurons or channels, the network can be more sparse and efficiency of networks can be improved. [Han *et al.*, 2015] proposes to prune the neural connections with small weights. [Li *et al.*, 2016] proposes to prune the channels with small weights and then fine-tune the network. [Yang *et al.*, 2016] proposes a layer-by-layer pruning algorithm by minimizing the error in the output features. [Luo *et al.*, 2017] prunes the channels according to the next layer's feature reconstruction error. [Yu *et al.*, 2017] propagates the feature ranking on the final response layer to obtain neuron importance scores. [Liu *et al.*, 2017] proposed to make use of the scaling factors in Batch normalization [Ioffe and Szegedy, 2015] for pruning channels. [Zhang *et al.*, 2018] formulates pruning as a constrained nonconvex optimization problem.

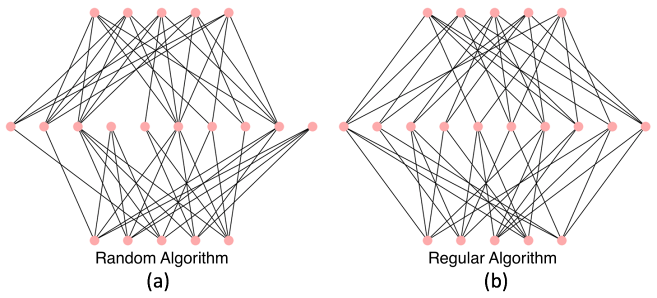

Figure 1: Example neural graphs construction using random algorithm and regular algorithm. The left graph is constructed by choosing the vertices on other side uniformly and independently at random, while the right one controls the regularity instead of random construction. It is obvious that random algorithm might block data flow due to its uncertainty, which regular algorithm can avoid.

Typical network pruning techniques focus on keeping important weights and fine-tune pruned models. However, recent works argue that the pruned architecture itself contributes to the final efficiency [Zhu and Gupta, 2017; Liu *et al.*, 2018]. Getting lost in manipulating individual neurons or channels, we could ignore the big picture of the neural network. To illustrate, we constructed two toy networks of 2 layers with same number of connections under different algorithms. The random algorithm is constructed by randomly selecting the neural connections in the neural graph, whereas the regular algorithm randomly selects the neural connections under the constraint of regularity. For example, poor regularity may block data flows and hinder neurons or channels from getting involved in the network, as shown in Figure 1a, which is generated by random algorithm. It is therefore necessary to have a thorough investigation on the characteristics displayed by the neural network as a whole (see Figure 1b), which forces all the vertices on the same side have similar degrees.

In this paper, we propose to craft efficient deep neural network through a graph lens. Structural complexity reveals the way in which vertices and edges are arranged in the graph, providing a significant influence on the graph function and performance. Graph entropy offers an attractive route to such complexity measures. To increase the capacity of the pruned network under a particular network sparsity, we maximize graph entropy of the network by optimizing the arrangements of neurons and connections. We identify important weights from the pre-trained over-parameterized network, and use them in preference to others in crafting our efficient neural network. Based on the resulting sparse network architecture, we train the network parameters from scratch rather than adapting their original weights. The proposed algorithm can be easily deployed to many popular network architectures, such as ResNet [He *et al.*, 2016], VGG networks [Simonyan and Zisserman, 2014] and DenseNet [Huang *et al.*, 2016]. Experimental results on ImageNet and CIFAR datasets [Krizhevsky, 2009; Deng *et al.*, 2009] demonstrate that deep neural networks can be well compressed by investigating graph entropy while preserving the accuracy.

2 Methodology

Pruning neural networks is to compresses networks by deleting neurons or neuron connections from a trained model, which has been paid more attention to in recent years. However, most pruning techniques only involve local operations and do not take whole network proprieties into consideration, which may block information flows from layer to layer. In contrast, we take neural network architecture as a graph, and construct sparse graphs with a global viewpoint to initialize the network architecture before the training phase. To encourage better information flow in the network, we employ Von Neumann Entropy as a measurement to assess the qualities of graphs, which leads to a favorable tradeoff between accuracy and sparsity of the neural network.

2.1 Von Neumann Entropy

Von Neumann Entropy is an extension of the Gibbs entropy to the quantum field, which can be treated as a quantitative measure of mixedness of density matrices [Braunstein *et al.*, 2004]. Recently, considering its capability of describing spectral complexity, centrality, and entanglement of the graph, Von Neumann Entropy has been further explored to evaluate graph entropy in various graph pattern recognition and analysis applications. The definition of Von Neumann Entropy is given as

$$S(\rho) = -tr(\rho \ln \rho), \tag{1}$$

where tr denotes the trace of matrix and ρ is the density matrix. ρ could be a Laplacian matrix L_G scaled by degree sum of graph G, i.e. $\rho = \frac{1}{d_G} L_G$. Given λ_i as the i-th eigenvalue of density matrix ρ, Von Neumann Entropy can be re-written in terms of spectrum of ρ as

$$S(\rho) = -\sum_{i=1}^{n} \lambda_i \ln \lambda_i. \tag{2}$$

The Shannon entropy computes the uncertainty of global spectral parameters of graph, involving all the eigenvalues, which makes it a useful and general measurement. Here we pay more attention to properties of Von Neumann Entropy.

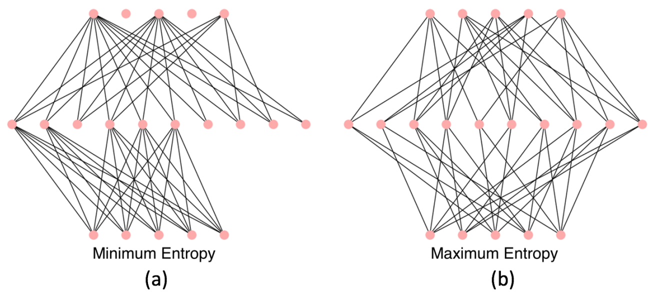

Figure 2: Example neural graphs construction using greedy algorithm. Given the same number of edges, the left graph is the one with minimum entropy and the right one with maximum entropy. We can see that a graph with more fully-connected clusters tends to have small entropy and a well-balanced one tends to have large entropy.

To illustrate, we first constructed two toy graphs with the minimum and maximum entropy using greedy algorithm to explore its properties, as shown in Figure 2. We observed that given a fixed number of edges, if there are more connected clusters that are disjoint unions of highly fully-connected subgraph, the graph will have a smaller entropy. This is consistent with the results in [Passerini and Severini, 2008]. The entropy of the graph in Figure 2 (a) is 2.554. Almost half of connections from the first layer to bottom layer have been blocked and several vertices are deactivated due to the minimum entropy construction. On the contrary, a "balanced" graph that has a higher regularity tends to have a larger entropy. A more rigorous proof will be given later. For example, we plot a graph that is constructed with the maximum entropy given 50 edges in Figure 2 (b) whose entropy is 2.875. All vertices on the same layer have similar degree, which produces a balanced graph and results in better connections and data flows. If a graph is more balanced, every neuron would be more active in contributing to the entire neural network, which results in a better network performance.

An efficient deep neural network is expected to have a better inner connection for data flows and high-sparsity for efficient compression, a graph of large entropy exactly tickes all the boxes. Consider the neural graph $G = (V, E)$, where V is the set of vertices with size n and E is the set of edges with size m. We can use greedy algorithm to maximize the entropy of graph. We simply compute graph entropy increment by adding all the possible edges and select the one which contributes the maximum increment. Algorithm 1 shows the details of it. After the neural graph construction, a network can be easily crafted from the graph.

For a linear layer, it can easily built by treating vertices as neurons and edges as neuron connections. For a convolutional layer, it can be built by a similar way where we treat vertices as channels and edges as filters. The number of connections or size of filters depends on hyper parameter m', which is the number of selected edges in Algorithm 1.

However, greedy algorithm often takes lots of time to craft graphs. To select an edge added to graph, we need to compute the Entropy increment for all the possible edges with size m and computation of Von Neumann Entropy takes $\mathcal{O}(n^3)$, so each step in greedy algorithm takes $\mathcal{O}(mn^3)$ and the total complexity becomes $\mathcal{O}(m^2n^3)$. Thus, we tried making use of properties of graph entropy to reduce the complexity.

Algorithm 1 Neural graph generation with greedy algorithm

```
Input: number of total edges $m$, number of edges to select $m'$
Output: graph $G$ with size of $m'$
Initialize graph $G = []$
Initialize edges $E$ with size $m$, which are edges of complete neural graph (fully connected)
repeat
    $Entropy_{max} = 0$
    for $i = 0$ To $i = |E| - 1$ do
        $Entropy(G \cup E_i) = Entropy of G \cup E_i$
        if $Entropy_{max} < Entropy(G \cup E_i)$ then
            $E_{max} = E_i$
            $Entropy_{max} = Entropy(G \cup E_i)$
        else
            Continue
        end if
    end for
    Add edge $E_{max}$ to $G$
until $|G|$ is $m'$
```

2.2 Regularity of the Graph

In graph theory, a graph is taken as regular, if its vertices have the same degree or valency. "Regularity" thus describes the extent of the graph to be regular, and can be computed as

$$R = -std(d), \tag{3}$$

where std denotes the standard deviation and d denotes the degrees of all vertices in the graph.

We found that regularity has a strong connection to Von Neumann Entropy. By approximating Von Neumann Entropy using quadratic entropy [Lin and Zhou, 2018], we have

$$S(\rho) \approx tr(\rho(I_n - \rho)), \tag{4}$$

where I_n is an identity matrix. Recall $\rho = \frac{1}{d_G} L_G$ and the degree sum of graph $d_G = 2m$. Eq. 4 can be rewritten as

$$\begin{aligned} S(\rho) &\approx tr(\frac{L_G}{2m}(I_n - \frac{L_G}{2m})) \\ &= \frac{1}{2m} tr(L_G) - \frac{1}{4m^2} tr(L_G^2). \end{aligned} \tag{5}$$

Based on the properties of Laplacian matrix that L_G is symmetric and the trace of it is equal to $2m$, Von Neumann Entropy approximation can be rewritten in a form of degree, computed as

$$S(\rho) \approx 1 - \frac{1}{2m} - \frac{1}{4m^2} \sum_{v \in V} d_v^2, \tag{6}$$

where d_v denotes the degree of vertex v. Given an edge (a, b) between vertices a and b, the increment in entropy of adding this edge to graph can be computed as

Algorithm 2 Random regular neural graph generation

```
Input, output and Initialization are same with Algorithm 1
Initialize $D$ with the size of $n$ to record the degrees
repeat
    For edge $[a, b]$ in $E$, compute the squared sum of degree $dS = (d_a + 1)^2 + (d_b + 1)^2$
    Find the edges with minimum $dS_{min}$ from $E$, marked as $E'$
    Randomly select one edge $[u, v]$ from $E'$
    Add edge $[u, v]$ to $G$
    Update $D_u$ and $D_v$ in $D$
until $|G|$ is $m'$
```

$$\begin{aligned} &S(\rho(G \cup (a, b))) - S(\rho(G)) \approx \\ &\frac{1}{2m} + \frac{1}{4m^2} \sum_{v \in V} d_v^2 - \frac{1}{2(m+1)} \\ &- \frac{1}{4(m+1)^2} (\sum_{v \neq a,b} d_v^2 + (d_a + 1)^2 + (d_b + 1)^2). \end{aligned} \tag{7}$$

From Eq. 7, it is obvious that the increment of entropy depends on d_a and d_b. Graph G can obtain more increment in entropy if vertices a and b have smaller degrees. We reduce the cost of searching, by simply choosing the vertices pair with smallest squared sum of degrees in each step, instead of computing the entropy increment of adding all the possible edges, which reduces computation complexity to $\mathcal{O}(m^2)$. Algorithm 2 shows the details of our regular construction. The complexity can be further reduced by exploring the vertex degree boundaries. The objective is to minimize the squared sum of degrees and we can derive a lower bound by making use of inequality as

$$(d_a + 1)^2 + (d_b + 1)^2 \geq \frac{(d_a + d_b + 2)^2}{2} \tag{8}$$

From Eq. 8, we obtain a sub-optimal solution by directly minimizing the lower bound $d_a + d_b$, which can be used to further decrease the complexity based on Proposition 1.

Proposition 1. *The degrees of all vertices are in range* $[\lfloor \frac{2m}{|V|} \rfloor, \lceil \frac{2m}{|V|} \rceil]$.

Proof. To prove it by contradiction, we assume the opposite. (a) If there exists a vertex X has degree of $\lceil \frac{2m}{n} \rceil + 1$, all other vertices have minimum degree of $\lceil \frac{2m}{n} \rceil$ or X cannot be selected. And the sum degree of graph will be $2m + 1$, which conflicts with the definition of graph. (b) If there exists a vertex X has degree of $\lfloor \frac{2m}{n} \rfloor - 1$, X has not been assigned to an edge added to the graph in the final step, which conflicts with the principle that we choose the vertices pair with smallest degrees. Thus all the degrees of vertices are in range $[\frac{2m}{n} - 1, \frac{2m}{n}]$. □

Based on proposition 1, high regularity is the property our target graph must have and we simply adopt random algorithm to generate a graph with high regularity by randomly adding edges to graph while restricting degree upper boundary of all the vertices, which reduces computation complexity

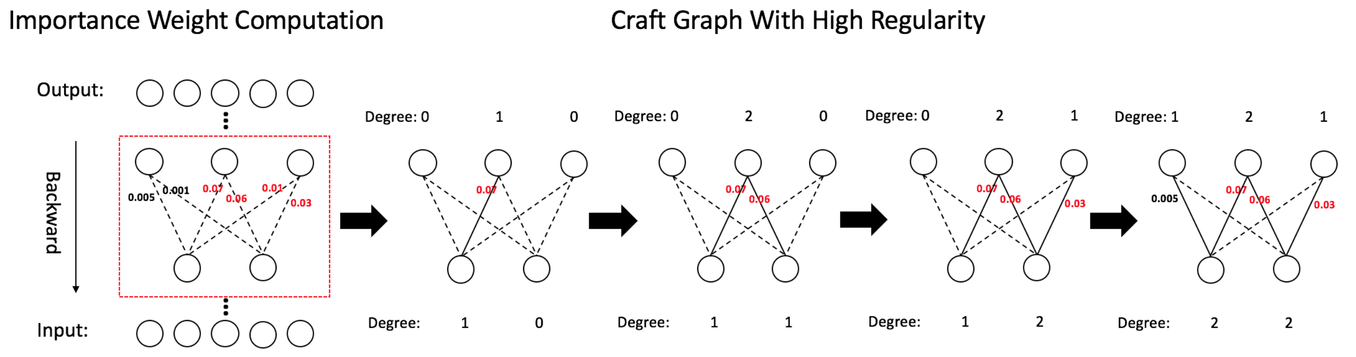

Figure 3: An illustration of our algorithm. Dotted lines denotes the complete neural graph and full ones denotes the edges added to the final sparse graph. With importance weights, we add edges with maximum importance scores one by one to the graph. Meanwhile, we force the regularity of graph. For example, the edges with red values are the ones with top importance scores in the layer, however, those added to graph sometimes are not exactly these edges due to the regularity.

to $O(m)$. In terms of models which contains massive layers and neural connections, we can simply divide the entire network into several subnetworks, which further reduces the complexity of construction.

2.3 Importance Weight

Although we reduce the construction complexity, the output graph is not fixed due to the random algorithm which randomly select the edge with minimum squared sum of degrees because there is no criterion for selection when the graph exists multiple vertices pairs with minimum sum, which makes the performance unstable, shown in Algorithm 2. Thus, we introduce importance weights which can be easily obtained to tackle this problem by giving the selection criterion.

Algorithm 3 Regular neural graph generation with importance weights

```
Input, output and Initialization are same with Algorithm 1
**Additional Input:** importance scores $S$
**Sort** $S$ in descending order
**Sort** $E$ according to $S$
**repeat**
    Select edge $[u, v]$ from $E$ in order
    **if** degree($u$) $< U_{max_d}$ and degree($v$) $< V_{max_d}$ **then**
        **Add** edge $[u, v]$ to $G$
    **else**
        **Continue**
    **end if**
**until** $|G|$ is $m'$
```

Importance Estimation By Gradient: The role of importance estimation is illustrated in Figure 3. Given the network N and input x, the output of network is $N(x;\theta)$, where θ denotes weight parameters of network. The neuron connections (for FC layer) or filters (for CNN layer) may have different levels of importance according to the sensitivities of output to the infinitesimal changes on them. The output difference under perturbation can be estimated by simply computing the gradients of them as

$$N(x;\theta+\epsilon) - N(x;\theta) \approx \sum_{i=1}^{k} \frac{\partial(N(x;\theta))}{\partial\theta_i}\epsilon_k, \qquad (9)$$

where ϵ is the perturbation on weight parameters and k is the number of parameters in the network. From Eq. 9, the sensitivity depends on the gradients of learned network with respect to the weight parameters on input x. Thus, we can obtain estimated importance weights by computing their gradients as

$$S(k) = \frac{1}{M}\sum_{i=1}^{M} \frac{\partial(N(x_i;\theta))}{\partial\theta_k}, \qquad (10)$$

where M is the number of examples from dataset. From Eq. 10, importance weights can be computed by M times backwards on a pre-trained model and selected dataset, which assists our initial sparse network to pay more attention to these connections with more important data flows.

With importance weights, we can construct the entire neural graph with high regularity by adding edges one by one according to their importance levels instead of random selection algorithm, shown in Figure 3. Algorithm 3 shows the details of our sparse regular graph construction. Thus regular algorithm with importance weights guarantees that edges which tend to have important data flow will be added to the graph, which makes the generated network adaptive to the specific dataset so that our network has more stable performance and gains better trade-offs.

Our final algorithm, regular algorithm with importance weights (RAIW) has taken both graph entropy and importance weights into consideration, which improves the efficiency due to graph entropy and guarantees the stability due to importance weights. The entire process of crafting neural graphs is shown in Figure 3. The edges with high importance will be added to our neural graph if it does not destroy the regularity of graph. For example, the edge with the top importance weights 0.01 cannot be added to our neural graph because it connects those vertices with higher degrees, thus the edge with value of 0.005 is added instead, shown in the final step in Figure 3.

3 Experiments

To evaluate the efficiency of our algorithms, we apply our RAIW algorithm to generate neural graphs based on different popular CNN architectures, such as VGG, Resnet and Densenet. For comparison, we repeat the experiments on different datasets, such as CIFAR10, CIFAR100 and Imagenet, with these architectures under various layer settings and compare them with pruning techniques or the original models to demonstrate better trade-offs of our algorithm. The stability and regularity will also be discussed.

3.1 Comparison With Efficient CNN Architectures And Pruning Algorithms

Techniques	Accuracy	Params
Original	94.0%	15.0M
[Li *et al.*, 2016]	93.4%	5.4M
[Liu *et al.*, 2017]	93.8%	2.3M
[Han *et al.*, 2015]	93.3%	1.5M
RAIW	93.4%	1.1M

Table 1: The performance of VGG16 network crafted by RAIW algorithm compared with original VGG16 and pruning techniques on CIFAR 10 dataset.

VGG on CIFAR10

We first compare our algorithm against two popular pruning techniques. [Han *et al.*, 2015] prunes the neural connections with small weights, [Li *et al.*, 2016] prunes the small incoming weights based on a pretrained model and regains the accuracy after the networks are fine-tuned and [Liu *et al.*, 2017] prunes the channels based on the scaling factors in Batch normalization and repeats the process of training, pruning and fine-tune. All of them achieve good performance among pruning technique. To compare with them, we evaluate on CIFAR10 [Krizhevsky, 2009] with VGG architecture [Simonyan and Zisserman, 2014].

The detailed results are shown in Table 1. Our algorithm can preserve the accuracy, which has 0.6% drop but only $1.1M$ parameters, almost $14X$ compression rate. Furthermore, our RAIW algorithm allows higher sparsity than $14X$ with a relatively small accuracy drop, which will be discussed in Section 3.3.

Densenet and Resnet on CIFAR100

To evaluate the robustness of our algorithm, we deploy RAIW on Densenet [Huang *et al.*, 2016] and Resnet [He *et al.*, 2016] running on CIFAR100 dataset [Krizhevsky, 2009]. We sparse these models by crafting convolutional layers whose filter size is half of the original one by controlling the number of selected edges in algorithms.

For Resnet on CIFAR100, we run Resnet with different number of layers (32, 56, 110) on CIFAR100 dataset. We compare our algorithm with these base models by comparing the 1-crop error along with the number of parameters of model, the details are given in Figure 4 (a). Our algorithm can consistently have a better performance with the similar number of parameters, comparing the two lines in Figure 4 (a). For example, the original Resnet-56 has 0.86M parameters number with 28.89% error, however, our RAIW algorithm which has the similar parameter number on Resnet-110, has 0.8% error drop.

For Densenet on CIFAR100, we run with Densenet-BC which contains bottleneck layers and uses Densenet-BC-40-24, 40-48, 40-60 which have 40 layers and different growth rates as base models. Again, we show better accuracy-parameters trade-offs, the details are given in Figure 4 (b). Similarly, comparing the two lines in Figure 4 (b), the network we crafted using RAIW algorithm can be more efficient. For example, RAIW algorithm has 21.07% error on Densenet-BC-40-60 with 2.10M parameters while the original Densenet-BC-40-48 has 21.37% error with 2.76M parameters, which demonstrates the efficiency of our algorithm.

Model	Accuracy	Params
RAIW-Resnet-50	70.4%	13.28M
RAIW-Resnet-101	**73.6%**	**21.08M**
Resnet-34	73.3%	21.78M
Resnet-50	75.3%	25.50M
RAIW-Densenet-169	71.5%	8.32M
RAIW-Densenet-201	72.4%	11.74M
Densenet-169	74.8%	13.99M
Densenet-201	75.6%	19.78M

Table 2: The accuracy performance of Resnet and Densenet crafted by RAIW evaluated on Imagenet dataset, compared with original architectures, ordered by number of parameters.

Densenet and Resnet on Imagenet

To further evaluate the efficiency and generality of our algorithm, we also test on Imagenet [Deng *et al.*, 2009] using the Resnet and Densenet architectures crafted by our algorithm.

For Resnet on Imagenet, we craft Resnet-50 and 101 and compare them to the original ones with similar parameter numbers. When the parameters number comes to 21M, our RAIW algorithm on Resnet-101 can gain a slightly better accuracy than Resnet-34 with less parameters, shown in Table 2 with bold. For Densenet on Imagenet, we craft Densenet-169, Densenet-201 and compare them to the original ones. For each Densenet model, RAIW has approximately 3% accuracy drop but only 40% parameters of the original one. Thus, RAIW can obtain better accuracy-parameters trade-offs and improve the efficiency of popular CNN architectures.

3.2 Stability Of Model

To evaluate the stability of models under constructions of different algorithms, we repeat the experiments on CIFAR10 with VGG16 architecture. For each algorithm, we repeat 5 times and compare their mean accuracy and standard deviation. We compare the regular algorithm 2 with RAIW 3 to demonstrate the role of importance weights. The results are shown in Table 3.

Algorithm	R1%	R2%	R3%	R4%	R5%	Average
Regular	92.69	92.66	92.97	92.70	93.06	92.82±0.17
RAIW	92.97	92.77	93.00	92.82	92.69	92.85±0.12

Table 3: VGG16 model under construction of regular algorithm and the one with importance weight over 5 runnings on CIFAR10 dataset. The final column "Average" denotes the mean accuracy ± standard deviation.

From the results, it is obvious that the random regular algorithm tend to have large variation on the accuracy due to the random initial construction. Although they force graphs to be regular to different extents, there still exists high uncertainty because different connection distributions always have different performance, which results in relatively high standard deviation, shown in the first row with 0.17. On the contrary, our RAIW algorithm has relatively high mean accuracy

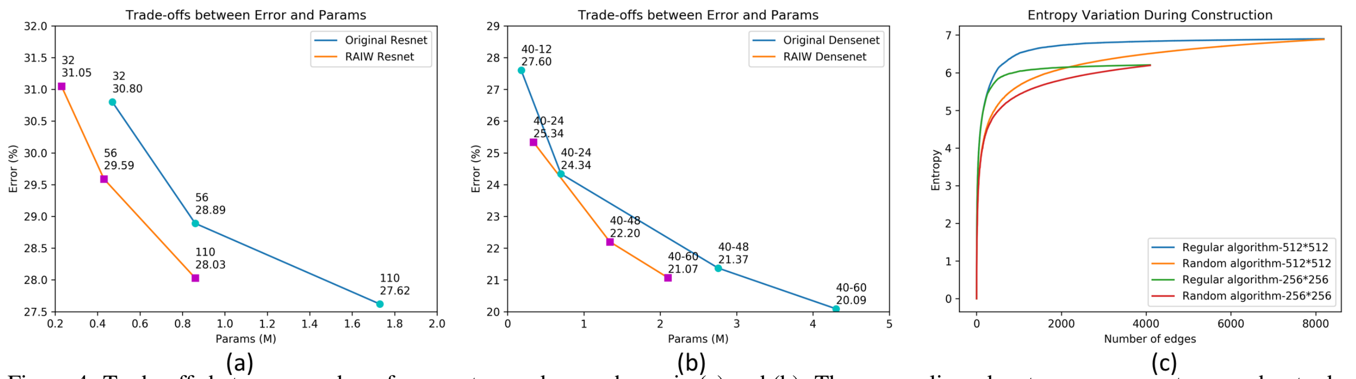

Figure 4: Trade-offs between number of parameters and error shown in (a) and (b). The orange lines denote error-parameters number trade-offs of our algorithm and the blue ones denote the original architecture. The two-line numbers on each data point in (a) denote the number of layers in ResNet on the first line and the corresponding error on the second line. Those in (b) denote the number of layers in DenseNet with its growth rate on the first line and the corresponding error on the second line. We show the performance of our algorithm applied to Resnet on CIFAR100 in (a), Densenet on CIFAR100 in (b). (c) illustrates entropy variation of 256*256 and 512*512 layers construction with degree of 16. From the line chart, regular algorithm can guarantee much faster entropy increment compared with random algorithm.

and low standard deviation with 92.85 ± 0.12 (shown in the last row) because once the importance weights are computed, the neural graph is fixed, which makes the performance more stable and the graph can be stored for multiple-times use, which saves the computation resources. As a result, RAIW algorithm improves the stability of regular algorithm, which makes it easily deployable.

3.3 Role Of Regularity

Due to regularity, d can be hyper parameters we use to sparsify the network, where d denotes the degrees of all the vertices. And accuracy relies heavily on it due to the trade-offs between sparsity and performance. In this section, we run VGG16 model on CIFAR10 under different regularity to evaluate trade-offs of our algorithms, as shown in Table 4.

VGG16	Original VGG	d-32 VGG	d-16 VGG	d-8 VGG
Conv1	3x64x3x3	3x64x3x3	3x64x3x3	3x64x3x3
Conv2	64x64x3x3	64x64x3x3	64x64x3x3	64x64x3x3
Conv3	64x128x3x3	64x128x3x3	64x128x3x3	64x128x3x3
Conv4	128x128x3x3	128x64x3x3	128x64x3x3	128x32x3x3
Conv5	128x256x3x3	128x32x3x3	128x16x3x3	128x8x3x3
Conv6	256x256x3x3	256x32x3x3	256x16x3x3	256x8x3x3
Conv7	256x256x3x3	256x32x3x3	256x16x3x3	256x8x3x3
Conv8	256x512x3x3	256x32x3x3	256x16x3x3	256x8x3x3
Conv9	512x512x3x3	512x32x3x3	512x16x3x3	512x8x3x3
Conv10	512x512x3x3	512x32x3x3	512x16x3x3	512x8x3x3
Conv11	512x512x3x3	512x32x3x3	512x16x3x3	512x16x3x3
Conv12	512x512x3x3	512x32x3x3	512x16x3x3	512x16x3x3
Conv13	512x512x3x3	512x32x3x3	512x16x3x3	512x16x3x3
Linear1	512x512	512x128	512x128	512x64
Linear2	512x10	512x10	512x10	512x10
Total	14.98M	1.07M	0.75M	0.55M
Accuracy	93.96%	93.37%	93.06%	91.85%

Table 4: Details of each layer and number of parameters with accuracy of VGG16 model under different sparsity.

We have tried 3 different sparsity, marked as $d-32, d-16, d-8$, and details of filter size or neural connection number of each layer are given in the Table 4. And the results show satisfactory trade-offs, $d-32$ gains 93.37% accuracy with 1.07M parameters, compared with original VGG16 architecture which gains 93.86% accuracy with 14.98M parameters, our crafted neural graph can obtain "14x" compression rate with only 0.6% accuracy drop. Furthermore, $d-16$ with "20x" compression rate and $d-8$ with "27x" compression rate, both have relatively small accuracy drop.

The reason why our constructed architecture can have high sparsity with less accuracy drop is that our algorithm increment on graph entropy is much faster than other ones, as shown in Figure 4 (c). For the simplicity, we construct two subnetworks and each of them contains neural connections of two layers, which are $256 * 256$ and $512 * 512$. The blue and green curves denote the entropy increment along with the increasing of edge number under our regular algorithm 2 while the orange and red ones denote random algorithm. It is obvious that our algorithm keeps obtaining large increment during the first 1000 edges construction for $256 * 256$ layer and during the first 2000 edges construction for $512 * 512$ layer. The graph entropy almost reaches its peak when we select $d = 8$ where number of edges are $256 * 8 = 2048$ for 256*256 layer and $512 * 8 = 4096$ for 512*512 layer. We found it interesting that $d = 8$ is exactly the sparsity we have selected for $d-8$ and that's the sparsity where we found the accuracy tends to have a relatively larger drop. Thus, we believe that the accuracy is highly related to entropy value. As a result, our regular algorithm guarantees maximum increment of entropy in each step so that it can gain large entropy with less edges, which enables our model to have high sparsity.

4 Conclusion

Notice the strong connection between nerual networks and graphs, we tried to improve the efficiency by construct neural graphs which have better vertices and connections arrangement. Thus, we propose to craft efficient neural networks based on the properties of graph entropy. We reduce the computation complexity of graph generation to make it deployable and make use of importance weights to guarantee stability. Our RAIW algorithm achieves better trade-offs compared to pruning algorithms and efficient CNN architectures.

Acknowledgement

This work was supported in part by the Australian Research Council under Project DE180101438.

References

[Bansal *et al.*, 2018] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? In *Advances in Neural Information Processing Systems*, pages 4261–4271, 2018.

[Braunstein *et al.*, 2004] Samuel Braunstein, Sibasish Ghosh, and Simone Severini. The laplacian of a graph as a density matrix: A basic combinatorial approach to separability of mixed states. *Annals of Combinatorics*, 10, 07 2004.

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

[Han *et al.*, 2015] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

[Hassibi and Stork, 1993] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 164–171. Morgan-Kaufmann, 1993.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[He *et al.*, 2017] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *CoRR*, abs/1707.06168, 2017.

[Huang *et al.*, 2016] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[Krizhevsky, 2009] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[LeCun *et al.*, 1990] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.

[Li *et al.*, 2016] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

[Lin and Zhou, 2018] Hongying Lin and Bo Zhou. On the von neumann entropy of a graph. *Discrete Applied Mathematics*, 247:448 – 455, 2018.

[Liu *et al.*, 2017] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *CoRR*, abs/1708.06519, 2017.

[Liu *et al.*, 2018] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *CoRR*, abs/1810.05270, 2018.

[Luo *et al.*, 2017] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *CoRR*, abs/1707.06342, 2017.

[Passerini and Severini, 2008] Filippo Passerini and Simone Severini. The von neumann entropy of networks. *University Library of Munich, Germany, MPRA Paper*, 12 2008.

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[Szegedy *et al.*, 2014] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[Wu *et al.*, 2018] Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. *CoRR*, abs/1806.09228, 2018.

[Yang *et al.*, 2016] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *CoRR*, abs/1611.05128, 2016.

[You *et al.*, 2017] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1285–1294, New York, NY, USA, 2017. ACM.

[Yu *et al.*, 2017] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. NISP: pruning networks using neuron importance score propagation. *CoRR*, abs/1711.05908, 2017.

[Zhang *et al.*, 2018] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic DNN weight pruning framework using alternating direction method of multipliers. *CoRR*, abs/1804.03294, 2018.

[Zhou *et al.*, 2016] Hao Zhou, Jose M. Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 662–677, Cham, 2016. Springer International Publishing.

[Zhu and Gupta, 2017] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. 10 2017.