SEARCHING FOR STAGE-WISE NEURAL GRAPHS IN THE LIMIT

Anonymous authors
Paper under double-blind review

ABSTRACT

Search space is a key consideration for neural architecture search. Recently, Xie et al. (2019a) found that randomly generated networks from the same distribution perform similarly, which suggests we should search for random graph distributions instead of graphs. We propose graphon as a new search space. A graphon is the limit of Cauchy sequence of graphs and a scale-free probabilistic distribution, from which graphs of different number of vertices can be drawn. This property enables us to perform NAS using fast, low-capacity models and scale the found models up when necessary. We develop an algorithm for NAS in the space of graphons and empirically demonstrate that it can find stage-wise graphs that outperform DenseNet and other baselines on ImageNet.

1 INTRODUCTION

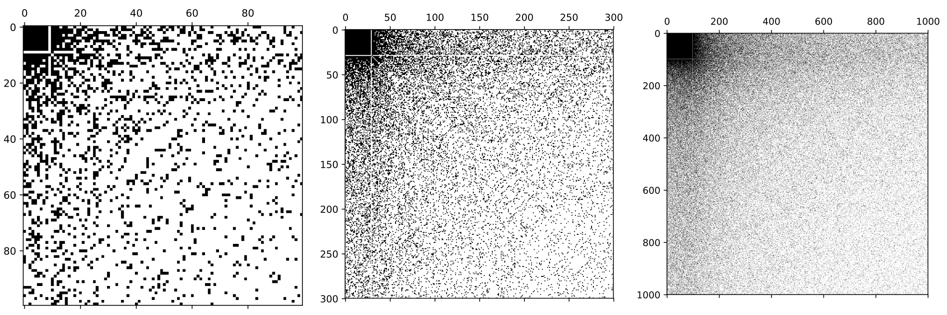
Neural architecture search (NAS) aims to automate the discovery of neural architectures with high performance and low cost (such as FLOPs or power consumption). Of primary concern to NAS is the design of the search space (Radosavovic et al., 2019), which needs to balance multiple considerations. For instance, too small a space would exclude many good solutions, whereas a space that is too large would be prohibitively expensive to search through. An ideal space should have a one-to-one mapping from parameter space to solutions and sufficiently smooth in order to accelerate the search.

A common technique (Zoph et al., 2018; Liu et al., 2018; Zhong et al., 2018; Liu et al., 2019b; Real et al., 2019) to keep the search space manageable is to search for a small cell structure, typically containing about 10 operations, each having 1-2 input sources. When needed, identical cells are stacked to form a large network. This trick allows cells found on, for instance, CIFAR-10 to work on ImageNet. Though this practice is effective, it cannot be used to optimize the overall network structure.

In both manual and automatic network design, the overall network structure is commonly divided into several stages, where one stage operates on one spatial resolution and contains several near-identical layers or multi-layer structures (i.e., cells). For example, ResNet-34 (He et al., 2016) contains 4 stages with 6, 8, 12, and 6 convolutional layers, respectively. DenseNet-121 (Huang et al., 2017) contains 4 stages with 6, 12, 24, and 16 layers. AmoebaNet-A (Real et al., 2019) has 3 stages, within each 6 cells of the same structure are arranged sequentially. Within each stage, most connections between layers are sequential with skip connections occasionally used. As an exception, DenseNet introduces connections between every pairs of layers within the same stage.

Here we emphasize the difference between a stage and a cell. A cell typically contains about 10 operations, each taking input from 1-2 other operations. In comparison, a stage can contain 60 or more operations organized in repeated patterns and the connections can be arbitrary. A network usually contains only 3-4 stages but many more cells. In this paper, we focus on the network organization at the level of stage rather than cell.

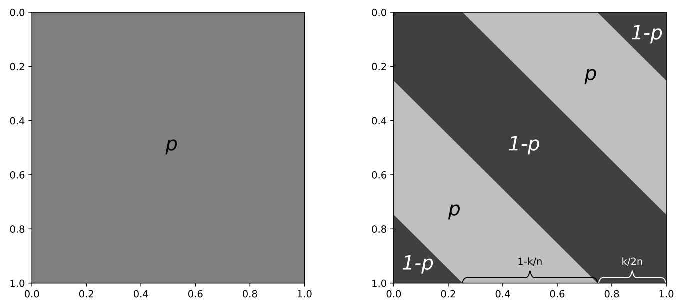
Xie et al. (2019a) recently showed that the stage structure can be sampled from probabilistic distributions of graphs, including Erdős-Rényi (ER) (Erdős & Rényi, 1960), Watts-Strogatz (WS) (Watts & Strogatz, 1998), and Barabási-Albert (BA) (Barabási & Albert, 1999), yielding high-performing networks with low variance in performance. This finding suggests the random graph distribution, rather than the exact graph, is the main causal factor behind network performance. Thus, searching


(a) $m_0 = m = 10$, $n = 100$

(b) $m_0 = m = 30$, $n = 300$

(c) $m_0 = m = 100$, $n = 1000$

Figure 1: Three adjacency matrices of graphs generated by the Barabási-Albert model with $m = m_0 = 0.1n$. A black dot at location (i, j) denotes an edge from vertex i to vertex j. The initial graph containing the m_0 vertices are set to be fully connected, though other setups are possible. The sequence of matrices converges to its limit, the graphon, as $n \to \infty$ when the ratios m_0/n and m/n remain unchanged.



(a) Erdős-Rényi

(b) Watts-Strogatz

Figure 2: Graphons for common random graph models. Different shades denote different probabilities (e.g., p and $1 - p$). The Erdős-Rényi model has two parameters: number of vertices n and probability p. The Watts-Strogatz (WS) model has three parameters: number of vertices n, replacement probability p, and initial neighborhood width k. Technically, the WS model has a constant number of edges, violating exchangeability for random graphs. Graphs sampled from (b) converges in probability to the same number of edges as n increases.

for the graph is likely not as efficient as searching for the random graph distribution. The parameter space of random graph distributions may appear to be a good search space.

We propose a different search space, the space of graphons (Lovász & Szegedy, 2004), which are limit objects of sequences of finite graphs, and argue that graphon is a superior search space than the parameter spaces of the random graph models. A graphon is a measurable function defined on $[0, 1]^2 \to [0, 1]$ and a probabilistic distribution from which graphs can be drawn. The space of graphons is a Cauchy completion of the space of finite graphs, under the so-called cut distance metric. We formally introduce the notion of graphon in Section 3.

Figure 1 visualizes three adjacency matrices randomly generated by the Barabási-Albert (BA) model with an increasing number of graph vertices. It is easy to see that, as the number of vertices increases, the sequence of random graphs converges to a limit, which is a graphon. The BA model starts with an initial seed graph with m_0 vertices and arbitrary interconnections. It sequentially adds new vertices until there are n vertices in the graph. For every new vertex, it adds m edges. The probability of adding an edge between the new vertex and an existing vertex v_i is proportional to the degree of v_i. In Figure 1, we let $m = m_0 = 0.1n$. The fact that different parameters result in the same adjacency matrix suggest that directly searching in the parameter space will revisit the same configuration and is less efficient than searching in the graphon space.

Additionally, graphon provides a unified and more expressive space than common random graph models. Figure 2 illustrates the graphons for the WS and the ER models. We can observe that these random models only capture a small proportion of all possible graphons. The graphon space allows new possibilities such as interpolation or striped combination of different random graph models. Novel structures (like those in NAS-Bench-101 (Ying et al., 2019)) likely do not follow any conventional random graph model but can be represented by graphons.

Finally, graphon is *scale-free*, so we should be able to sample an arbitrary-sized stage-wise architecture with identical layers (or cells) from a graphon. This allows us to perform expensive NAS on small datasets (e.g., CIFAR-10) using low-capacity models and obtain large stage-wise graphs to build large models for complex datasets like ImageNet.

In practice, searching in the graphon space requires special care. Since the limit object is never observed in reality, we are forced to approximate it with finite means. We propose an algorithm for searching for graphons that represent good designs of stage-wise neural architecture. We empirically demonstrate that sampling larger networks for ImageNet from small networks found on CIFAR-10, exploiting the scale-free property. The resultant network outperforms DenseNet, a parameter-efficient network with complex stage structure, using slightly fewer parameters.

The contributions of the paper are as follows:

- We propose graphon as a search space for stage-wise neural architecture that consists of connections among mostly identical units. Graphon generalizes random graphs. We point out some nice theoretical properties of graphon that are beneficial for the search of stage-wise architectures.
- We empirically demonstrate that by exploiting the scale-free property of graphon, we can scale the stage-wise architecture found on CIFAR-10 to a suitable size for ImageNet. The networks we found outperform DenseNet with slightly fewer parameters.

2 Related Work

We review the NAS literature with a focus on the distinction between cell and stage structures. In the pioneering work of Zoph & Le (2017), a recurrent neural network served as the controller that outputs all network parameters for all layers without such distinction. Later works attempted to reduce the heavy cost of search by constraining the search to cell structures only. Zhong et al. (2018) searched for a single cell structure and perform downsampling using pooling operations. Zoph et al. (2018), Xie et al. (2019b), and Real et al. (2019) searched for two types of cells: a reduction cell that includes downsampling, and a normal cell that does not. Liu et al. (2018) grew the cell structure from the simplest 1-operation cell to the maximum of 5 operations. Downsampling was performed by the same cell with stride 2. DARTS (Liu et al., 2019b) relaxed discrete choices in a cell to real numbers between 0 and 1, enabling gradient-based optimization. Casale et al. (2019) imposed a probabilistic formulation on top of DARTS.

Perhaps inspired by manual design of stage structures (e.g., Huang et al. 2017; Larsson et al. 2017), searching for higher levels of neural architecture has received increasing interest. Xie et al. (2019a) found that random graphs generated by properly parameterized Watts-Strogatz models outperform manual designs for stage-wise structures. Wortsman et al. (2019) redefined the stage-wise graphs so that a vertex can take multiple inputs but output only a single channel, resulting in large stage-wise graphs with up to 2560 vertices. Liu et al. (2019a) optimized the global structure of downsampling and upsampling for semantic segmentation. Ryoo et al. (2019) evolved the connections between multiple residual blocks for applications in video processing. Greff et al. (2017) analyzed ResNet and the Highway network and suggested that the purpose of layers within the same stage is to iteratively refine features and reduce variance. This finding hints at reusing stage structures that are effective at feature refinement. In summary, we believe that NAS for stage structures is still at an early stage with many unexplored opportunities.

Another approach for accelerate search is to share weights among different architectures. Saxena & Verbeek (2016) built a lattice where a chain-structured network is a path from the beginning to the end. In ENAS (Pham et al., 2018), a controller is trained using gradient policy to select a subgraph, which is subsequently trained using cross-entropy. The process repeats, sharing network weights across training sessions and subgraphs. In the one-shot approach (Brock et al., 2017; Bender et al., 2018; Chen et al.), the hypergraph is only trained once. After that, subgraphs are sampled from the hypergraph and evaluated. Finally, the best performing subgraph is retrained. Our focus in this paper is to validate that the mathematical theory of graphon can be effectively operationalized and leave weight sharing as future work.

3 Background on Graphon

Definition 1. *A graphon is a symmetric, measurable function* $W : [0,1]^2 \to [0,1]$.

The definition of graphon is straightforward, but its relation with graphs requires some standard concepts from real analysis. **A metric space** is a space with a distance function. More specifically, a metric space is a set $\mathbb{M}$ with a metric function $d : \mathbb{M} \times \mathbb{M} \to \mathbb{R}$. d defines the distance between any two points in $\mathbb{M}$ and satisfies the following properties:

- $d(x, x) = 0$ (Identity).
- $d(x, y) = d(x, y)$ (Symmetry).
- $d(x, y) \leq d(x, z) + d(z, y)$ (Triangular Inequality).

As an example, the set of real numbers $\mathbb{R}$ with the absolute difference metric $d_{abs}(x, y) = |x - y|$ is a metric space. **A Cauchy sequence** is defined as a sequence $(x_1, x_2, \ldots)$ whose elements become infinitely close to each other as we move along the sequence. Formally, for any positive $\epsilon \in \mathbb{R}$, there exists a positive integer $N \in \mathbb{Z}_+$ such that $d(x_j, x_i) < \epsilon, \forall i, j \in \mathbb{Z}_+, i, j > N$. **A complete metric space** is a metric space $\mathbb{M}, d$ in which every Cauchy sequence converges to a point in $\mathbb{M}$. For instance, the metric space of rational numbers $\mathbb{Q}$ and the absolute value metric d_{abs} is not complete; the sequence defined by $x_0 = 1, x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}$ converges to an irrational number $\sqrt{2}$. The metric space of real numbers with d_{abs}, however, is a complete metric space.

In plain terms, working in a complete metric space assures us that familiar operations like taking the limit would not lead into the realm of unknowns. Interested readers are referred to the textbook by Sutherland (2009). Given any metric space, we can form its completion, in general, by adding limit points. In the case of graphs, the limits allow explicit descriptions.

Consider simple graphs $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, which are unweighted, and undirected graphs without loops or multiple edges. Borgs et al. (2008) show the following.

Theorem 1. *Every Cauchy sequence of simple graphs in the metric* $\delta_\Box$ *converges to a graphon.*

Theorem 2. *An arbitrary graphon is the limit of some Cauchy sequence in the metric* $\delta_\Box$.

Without excessive technical detail, we illustrate the metric $\delta_\Box$ for the case where two graphs have equal number of vertices. For a partition $\mathbb{S}, \mathbb{T}$ of $\mathbb{V}$, $\mathbb{S} \cup \mathbb{T} = \mathbb{V}, \mathbb{S} \cap \mathbb{T} = \varnothing$, we define $e_{\mathcal{G}}(\mathbb{S}, \mathbb{T})$ to be the number of edges between $\mathbb{S}$ and $\mathbb{T}$, or the size of the cut. The cut distance $\delta_\Box$ for two graphs $\mathcal{G}$ and $\mathcal{G}'$ with equal number of vertices is defined as

$$\delta_\Box(\mathcal{G}, \mathcal{G}') = \max_{\mathbb{S}, \mathbb{T} \subset \mathbb{V}} \frac{1}{|\mathbb{V}|^2} \left| e_{\mathcal{G}}(\mathbb{S}, \mathbb{T}) - e_{\mathcal{G}'}(\mathbb{S}, \mathbb{T}) \right| \tag{1}$$

The cut distance is a sensible metric for random graphs. Consider two random graphs with n vertices and edge density $\frac{1}{2}$. If we measure their distance as the number of edges we have to change to make them identical, the two graphs will appear quite distant. However, since they are drawn from the same distribution, we expect their distance to be small. In this case, the cut distance $\delta_\Box$ is, with high probability, only $O(1/n)$. For graphs with different numbers of vertices, a fractal mapping between the vertices is used. See Borgs et al. (2008) for more details.

Besides obvious utility in graph theory and statistics, graphon has attracted broad research interest. It is an effective tool for understanding asymptotic graph properties like motif counts or the degree distribution. In machine learning, graphon has found applications in hierarchical clustering (Eldridge et al., 2016) and graph classification (Haupt et al., 2017).

For directed graphs with the potential of self-loops, Diaconis & Janson (2008) define **a digraphon** as a 5-tuple $(W_{00}, W_{01}, W_{10}, W_{11}, w)$. For vertices i and j, $W_{00}(i, j)$ describes the probability that

Algorithm 1 Search for Graphon

```
1: procedure INITIALIZE($\mathbb{V}$, $k$)
2:     Input: totally ordered vertices $\mathbb{V}$, number of subsets $k$
3:     Output: $\mathbb{I}^s(v)$, $\boldsymbol{\theta}_{u,i}$, $\boldsymbol{\alpha}_v$, $\mathbb{U}_i$
4:     for each vertex $v \in V$ do
5:         $\mathbb{I}(v) \leftarrow \{u \mid u \prec v, u \in \mathbb{V}\}$    ▷ all vertices preceding $v$ to avoid cycles
6:         $\mathbb{I}^p(v) \leftarrow \mathcal{P}(\mathbb{I}(v))$    ▷ the powerset of $\mathbb{I}(v)$
7:         $\mathbb{I}^s(v) \leftarrow$ a random $k$-element subset of $\mathbb{I}^p(v)$
8:         $\boldsymbol{\alpha}_v \leftarrow$ a random vector in $\mathbb{R}^k$
9:         for each input subset $\mathbb{U}(v,i) \in \mathbb{I}_s(v)$ do
10:            for each vertex $u \in \mathbb{U}(v,i)$ do
11:                INITWEIGHT($\boldsymbol{\theta}_{u,i}$)    ▷ create a set of weights $\theta_{u,i}$ for $u$
12: procedure STAGEFORWARD($\mathbb{V}, \boldsymbol{x}, \tau$, $\mathbb{I}^s(v)$, $\boldsymbol{\theta}_{u,i}$, $\boldsymbol{\alpha}_v$, $\mathbb{U}_i$)
13:     Input: vertices $\mathbb{V}$, input $\boldsymbol{x}$, temperature $\tau$, initialized $\mathbb{I}^s(v)$, $\boldsymbol{\theta}_{u,i}$, $\boldsymbol{\alpha}_v$, $\mathbb{U}_i$
14:     Output: a feature map extracted by the current stage
15:     $in(v_{in}) \leftarrow \boldsymbol{x}$    ▷ $v_{in}$ is a dummy sending outputs to all vertices.
16:     for each vertex $v \in V$ do
17:         for each input subset $\mathbb{U}(v,i) \in \mathbb{I}_s(v)$ do
18:             for each vertex $u \in \mathbb{U}(v,i)$ do
19:                 $out(u,i) \leftarrow f(in(u), \boldsymbol{\theta}_{u,i})$    ▷ apply the operation of $u$ to its input
20:             $out(i) \leftarrow$AGGREGATE($out(u,i), \forall u$)    ▷ sum or concatenation
21:         Sample $\boldsymbol{\beta} \sim$ Gumbel(0, 1)
22:         $a_i \leftarrow \frac{\exp((\log \alpha_{v,i})+\beta_i)/\tau}{\sum_j \exp((\log \alpha_{v,j})+\beta_j)/\tau}$    ▷ perform Gumbel softmax
23:         $in(v) \leftarrow \sum_i a_i \, out(i)$
        return $in(v_{out})$    ▷ $v_{out}$ is a dummy whose input is the stage's output.
```

no edge exists between them; $W_{01}(i,j)$ describes the probability that an edge goes from i to j; $W_{10}(i,j)$ describes the probability that an edge goes from j to i, and $W_{11}(i,j)$ the probability that two edges go both ways. w is the probability for self-loops.

Connections within a feed-forward neural network are modeled as a directed acyclic graph. This can be realized by putting a total ordering $\prec$ on the vertices and requiring the following: $w = 0$ (no self-loops), $W_{11} = 0$ (no mutually connected vertices) and $W_{10}(i,j) = 0$ if $i \prec j$. In practice, we adopt a condensed representation using an upper-triangular square and define $W : [0,1]^2 \to [0,1]$ subject to $W(i,i) = 0$ and $W(i,j) = 0$ if $i > j$. That is, we only allow directed edges going from vertex i to vertex j if $i < j$; self-loops are disallowed.

4 SEARCH ALGORITHM

We now introduce the search algorithm. We first discuss how we let different graph configurations, represented by adjacency matrices, compete with each other. After that, we explain how we estimate the graphon and sample from it.

4.1 SEARCHING FOR AN ADJACENCY MATRIX

Graphs are discrete objects that cannot be optimized directly. It is common to use the Gumbel softmax trick (Jang et al., 2017; Wu et al., 2019) to allow gradient to flow back through discrete choices. Given a multinomial distribution with probabilities $\pi_0, \ldots, \pi_k$, we independently draw a k-dimensional vector $\boldsymbol{\beta}$ from a Gumbel distribution: $\forall i, \beta_i \sim$ Gumbel$(0,1)$. The Gumbel softmax is defined as $a_i = \exp\left((\log \pi_i) + \beta_i)/\tau\right) / \sum_j \exp\left((\log \pi_j) + \beta_j)/\tau\right)$. Empirically, we find it important to let different cells in the same stage learn to collaborate during the search. Therefore, we devise an algorithm that pits different input edge *combinations* against each other, instead of pitting one edge against no edge.

Algorithm 1 shows the detailed procedures. Specifically, for every vertex v on the graph, we sample k subsets of vertices that could provide input to v, while avoiding cycles (lines 5-7). For example, for

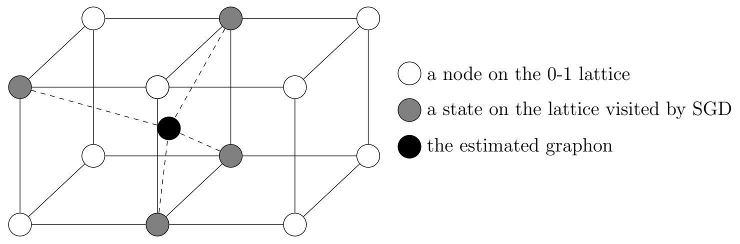

a node on the 0-1 lattice
a state on the lattice visited by SGD
the estimated graphon

Figure 4: The intuition on the search for the optimal graphon. When searching in the space of adjacency matrices, we only can only move on the 0-1 lattice. The optimal graphon, denoted by the filled black dot, is estimated by taking the average of the states visited by SGD.

the seventh vertex v_7, we could sample $\{v_1, v_3, v_5\}$ or $\{v_4, v_6\}$ and so on. The weights given to the different input subsets are denoted by $\boldsymbol{\alpha}$ (line 8). The model parameters for vertex u is denoted by $\boldsymbol{\theta}_u$. For every sampled input subset, we create separate parameters (lines 9-11), so that all vertices in the same subset can be trained to collaborate. This would be difficult if the parameters were shared across input subsets. During the forward computation, outputs from vertices in the same input subset are aggregated by either summation or concatenation (line 20). Finally, different input subsets are forced to compete by the Gumbel softmax (lines 21-23). After training, we pick the subset with the highest α as the winning configuration.

4.2 Estimating the graphon

Standard cross-entropy loss is used in the search. We use standard stochastic gradient descent (SGD) with momentum to optimize the model parameters $\boldsymbol{\theta}$ together with subset weights $\boldsymbol{\alpha}$. We perform the search for T_0 epochs with decreasing learning rate and decreasing Gumbel temperature τ. After that, we keep both the learning rate and temperature fixed and let the training run for additional T_1 epochs, each epoch producing a winning adjacency matrix. We take the average of the T_1 matrices as the estimated graphon.

This procedure has an intuitive explanation. When we search for the stage-wise graph, every entry in the adjacency matrix is either 0 or 1 (since there is either an edge or no edge between any two vertices). That is, we can only move along a 0-1 lattice. However, the graphon exists in continuous space and off the lattice. Figure 4 visualizes this intuition. Following the work of Mandt et al. (2016) and Izmailov et al. (2018), we assume that SGD with a constant learning rate during the last phase of training is equivalent to a Markov chain that draws samples from a stationary distribution, which is the distribution of random graphs conditioned on the graphon. We then adopt the maximum likelihood estimate to recover the graphon by taking the average of the matrices visited by SGD.

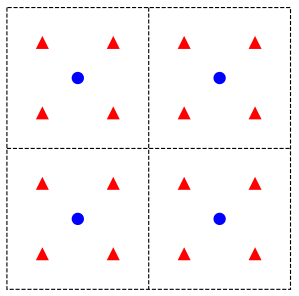


Figure 3: The estimated graphon contains sparse points represented by blue circles. Sampling an adjacency matrix twice as large requires the points located at the red triangles. In block-based smoothing, all points in the same small square take the same value.

4.3 Sampling from and scaling a graphon

Given a graphon, or a measurable function W which describes the density of edges, sampling a graph with n vertices is to sample every edge independently. We first find the corresponding point on the graph and draw from the binomial distribution (Chapter 10, Lovász 2012). For example, for an $n \times n$ matrix, the probability of the edge (i, j) is located at $W(\frac{i+0.5}{n}, \frac{j+0.5}{n})$. We can sample a binomial variable accordingly.

However, this procedure cannot be directly applied when we want to scale up or scale down the adjacency matrix. This is because our estimate a graphon from adjacency matrices only contain sparse points at $(\frac{i+0.5}{n}, \frac{j+0.5}{n})$. Figure 6 provides a visualization. In order to estimate the density of other points, certain smoothness assumptions must be made. The Regularity Lemmas indicate arbitrary graphons can be closely approximated by block models (Chapter 9, Lovász 2012). Thus, we adopt a block-based smoothing scheme by treating all values in any of $n \times n$ blocks in $[0, 1]^2$ as identical. For points with no known value in their block center, we opt for the nearest neighbor in the vertical direction.

5 EXPERIMENTS

5.1 SETUP

DenseNet (Huang et al., 2017) is a classical example where the stage-wise structure plays a major role in improving the network performance. Therefore, in the experiment, we focus on improving and comparing with DenseNet.

In order to create fair comparisons, we focus the search on the stage-wise structure and strictly follow DenseNet for the global network layout and the cell structure. The DenseNet network contains a stem network before the first stage, which contains a 3×3 convolution, batch normalization, ReLU and max-pooling. This is followed by three stages for CIFAR-10 and four stages for ImageNet. Between every two stages, there is a transition block containing a 1×1 convolution for channel reduction and a 2×2 average pool with stride 2 for downsampling. The network ends with a 7×7 global average pooling and a linear layer before a softmax.

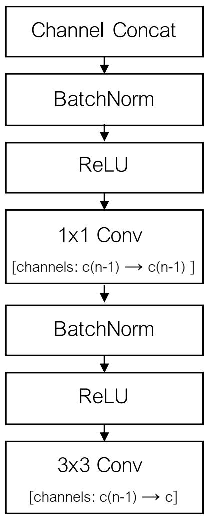


Figure 5: The DenseNet cell structure used in all experiments.

Figure 5 shows the cell structure for DenseNet, which contains two convolutions with different kernel size: 1×1 and 3×3. Each of the two convolutions are immediately preceded by a batch normalization and a ReLU activation function. Every cell in the same stage outputs c channels. The input to the n^{th} cell is the concatenation of outputs from the cell 1 to cell $n - 1$, for a total of $c(n - 1)$ channels. As every cell increments the number of input channels by c, it is called the growth rate.

As there is little prior research in the graphon space, we start the search from the graphon that corresponds to the WS distribution with k/n = 0.4 and $p = 0.75$, which we denote as WS-G(0.4, 0.75). We limit the search to input subsets that differ by 1 edge from the starting point. We perform the search on CIFAR-10 for 300 epochs with 8 cells in every stage. The search completes in 24 hours. For ImageNet training, we randomly draw, from the best graphon found by the search, four stage-wise graphs containing 8, 16, 32, and 16 cells, respectively by scaling up technique described in Section 4.3.

Since the search changes the stage structure, we adjust the growth rate c so that the numbers of parameters of all models are as close to DenseNet as possible to enable fair comparisons. We strictly follow the standard hyperparameters and data augmentation techniques used by DenseNet. For a detailed list of hyperparameters and data augmentation, see Appendix A.

Besides the original DenseNet, We introduce two baseline models. In the first model, the stage-wise graphs are generated by randomly deleting edges from the fully connected graph of DenseNet. In the second model, we use random graphs generated from WS-G(0.4, 0.75), which are similar to the best random architectures in Xie et al. (2019a). Both baselines contain the same number of cells as our method. The growth factors of each stage are manually adjusted so that the number of parameters is as close to DenseNet as possible.

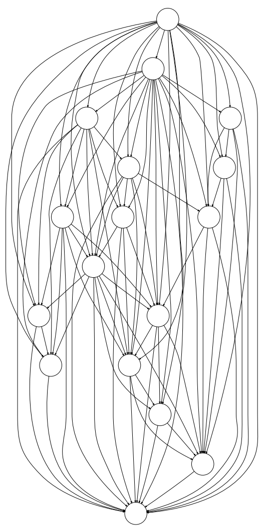


Figure 6: A 16-vertex graph (with a dummy input vertex) sampled from the graphon found by the search.

Table 1: Test performance on ImageNet and the number of parameters for all models.

Model	**Parameters**	**Top-1 (%)**	**Top-5 (%)**	**Top-1 10-crop (%)**	**Top-5 10-crop (%)**
DenseNet-121	7.978 M	75.01 ± 0.18	92.27 ± 0.05	76.56 ± 0.12	93.30 ± 0.05
Random Edge Deletion	8.079 M	74.79 ± 0.13	92.12 ± 0.12	76.39 ± 0.07	93.12 ± 0.07
WS-G(0.4, 0.75)	8.165 M	75.11 ± 0.02	92.28 ± 0.04	76.73 ± 0.12	93.30 ± 0.05
NAS Graphon (Ours)	7.933 M	75.18 ± 0.04	92.37 ± 0.05	76.79 ± 0.13	93.32 ± 0.03

In order to mitigate the effects of random noise, we run every model 5 times and report the average accuracy and standard deviation.

5.2 Results and Discussion

Table 1 shows the accuracy averaged over five runs and the standard deviation for all models. We make the following observations: First, the graphon found by our algorithm achieves the highest top-1 accuracy at 75.18% and the highest top-5 accuracy at 92.37%, despite having the least number of parameters. The NAS Graphon model has about 45K less parameters than DenseNet-121. The WS-G model achieves the second-best, albeit using 187K parameters more than DenseNet-121. Since we have strictly applied the same setting, including the global network structure and the cell structure, the difference in performance can only be attributed to the stage-wise graphs. These results strongly suggest (1) graphon provides an effective search space for NAS, and (2) the procedure for scaling up the graphon estimated from small graphs can find well-performing networks with increased capacity.

6 Conclusions

The design of search space is of paramount importance for neural architecture search. Recent work (Xie et al., 2019a) suggests that searching for random graph distributions is an effective strategy for the organization of layers within one stage. Inspired by mathematical theories on graph limits, we propose a new search space based on graphons, which are the limits of Cauchy sequences of graphs based on the cut distance metric.

We explain why graphon is a superior search space than the parameter space of random graph models such as the Erdős-Rényi model. Furthermore, we developed an algorithm for searching in the graphon space and empirically verify that it can indeed find effective stage-wise graphs for the classical DenseNet network. This work is a first step toward advanced NAS algorithms in the space of the compact and expressive representation of graphons.

References

Albert-Lászl Barabási and Réka Albert. Emergence of scaling in random networks. 286:509–512, 1999.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*. 2018.

Christian Borgs, Jennifer T Chayes, László Lovász, Vera T Sós, and Katalin Vesztergombi. Convergent sequences of dense graphs i: Subgraph frequencies, metric properties and testing. *Advances in Mathematics*, 219(6):1801–1851, 2008.

Andrew Brock, Theodore Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search. *CoRR*, abs/1902.05116, 2019. URL `http://arxiv.org/abs/1902.05116`.

Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. *arXiv preprint arXiv:1903.10979*.

P. Diaconis and S. Janson. Graph limits and exchangeable random graphs. *Rendiconti di Matematica e delle sue Applicazioni*, 28(1):33–61, 2008.

Justin Eldridge, Mikhail Belkin, and Yusu Wang. Graphons, mergeons, and so on! In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2307–2315. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6089-graphons-mergeons-and-so-on.pdf`.

P. Erdős and A Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pp. 17–61, 1960.

Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *ICLR*, 2017.

Andreas Haupt, Mohammad Khatami, Thomas Schultz, and Ngoc Mai Tran. Classification on large networks: A quantitative bound via motifs and graphons. *arXiv Preprint*, arXiv 1710.08878, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

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

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *UAI*, 2018.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commununications of the ACM*, 60(6):84–90, 2017.

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

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*. 2018.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*. 2019a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019b.

László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.

László Lovász and Balázs Szegedy. Limits of dense graph sequences. Technical Report Technical Report TR-2004-79, Microsoft Research, 2004.

Stephan Mandt, Matthew D. Hoffma, and David M. Blei. A variational analysis of stochastic gradient algorithms. In *ICML*, 2016.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4095–4104, 2018.

Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollr. On network design spaces for visual recognition. In *ICCV*, 2019.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI*. 2019.

Michael S. Ryoo, AJ Piergiovanni, Mingxing Tan, and Anelia Angelova. Assemblenet: Searching for multi-stream neural connectivity in video architectures. *arXiv Preprint. arXiv 1905.13209*, 2019.

Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *NIPS*. 2016.

Wilson A Sutherland. *Introduction to metric and topological spaces*. Oxford University Press, 2009.

D. J. Watts and S. H. Strogatz. Collective dynamics of ”small-world” networks. 393:440–442, 1998.

Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. 2019.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, pp. 10734–10742, 2019.

Saining Xie, Alexander Kirillov, Ross B. Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *ICCV*, 2019a.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *ICLR*, 2019b.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. 2019.

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

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

Barret Zoph, Vijay Vasudevan, Jonathan Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

A IMPLEMENTATION DETAILS

Some detailed hyperparameters in our experiments are as follows.

During the architecture search stage, we train for 300 epochs. The learning rate schedule follows DenseNet, i.e. The initial learning rate is set to 0.1, and is divided by 10 at epochs 150 and 225. Momentum is set at 0.9. Our batch size is 64 and growth rate in the DenseNet framework is set to 32. For the Gumble softmax, initial temperature is set at 1.0 and the minimum temperature is set at 0.1 with an anneal rate of 0.03.

For ImageNet training, we train for 90 epochs. We use label smoothing, which assigns probability 0.1 to all labels other than the ground truth. We use a batch size of 256 = 64 per GPU $\times$ 4 GPUs or 260 = 52 per GPU $\times$ 5 GPUs, depending on the GPU memory size. For the 4 stages, growth rates are set at 26,26,26,32 to match the number of parameters for Densenet-121. Learning rate is set at 0.1, divided by 10 at epochs 30, 60, and 80. We use Nesterov momentum of 0.9 and weight decay of 0.00005.

We also adopt the following data augmentation for ImageNet training, which are executed sequentially: Inception-style random cropping and aspect ratio distortion, resizing the image to the size 224 $\times$ 224, color jitter with factors for brightness, contrast and saturation all set to 0.4, horizontal flip with probability 0.5, and AlexNet-style random changes to the brightness of the RGB channels (Krizhevsky et al., 2017).