Neural Graph Embedding for Neural Architecture Search

Wei Li,[1] **Shaogang Gong,**[1] **Xiatian Zhu**[2]
[1]Queen Mary University of London, [2]University of Surrey
{w.li, s.gong}@qmul.ac.uk, xiatian.zhu@surrey.ac.uk

Abstract

Existing neural architecture search (NAS) methods often operate in discrete or continuous spaces *directly*, which ignores the *graphical topology knowledge* of neural networks. This leads to suboptimal search performance and efficiency, given the factor that neural networks are essentially *directed acyclic graphs* (DAG). In this work, we address this limitation by introducing a novel idea of *neural graph embedding* (NGE). Specifically, we represent the building block (i.e. the cell) of neural networks with a neural DAG, and learn it by leveraging a Graph Convolutional Network to propagate and model the intrinsic topology information of network architectures. This results in a generic neural network representation integrable with different existing NAS frameworks. Extensive experiments show the superiority of NGE over the state-of-the-art methods on image classification and semantic segmentation.

Introduction

Neural Architecture Search (NAS) is able to automate the tedious process of designing neural network architectures optimal for target tasks, bypassing the demand for rich domain knowledge and experiences. Recent attempts in NAS have achieved enormous success in various challenging tasks, e.g. image classification (Zoph and Le 2017), object detection (Ghiasi, Lin, and Le 2019), and semantic segmentation (Liu et al. 2019; Nekrasov et al. 2019).

There are three common learning paradigms among existing NAS methods: reinforcement learning (RL), evolutionary algorithm (EA), and gradient differentiable (GD) optimisation. A RL-based NAS method constructs a network architecture through deriving a sequence of discrete actions (each selecting an operator connection token), and uses its dev set accuracy as the reward. In a EA-based method, mutations and combinations of architectural elements are used to generate and search the architectures, where those architectures with higher fitness score (accuracy) are often selected to continue the evolution. Despite their remarkable performance, RL-based and EA-based NAS methods suffer from extremely low efficiency and high computational resource demand. For instance, to search a state-of-the-art architecture for CIFAR-10 and ImageNet, it takes 2000 GPU days for RL (Zoph and Le 2017) and 3150 GPU days for EA (Real et al. 2019). Several recent attempts have been made to improve, e.g. structural search space designing (Liu et al. 2018b; 2018a), architecture weights sharing and inheritance (Pham et al. 2018; Cai et al. 2018a; 2018b). Due to the fundamental searching challenge in a discrete space, RL and EA remain inefficient for NAS.

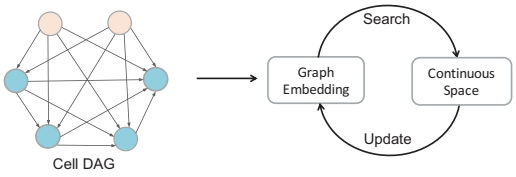

Figure 1: The concept of neural graph architecture search. We represent the cell of a network architecture with *directed acyclic graph* (DAG), which enables the search space to be represented in a continuous space, and facilitates the adoption of gradient descent based optimisation.

In contrast, a GD-based NAS method operates in a *continuous* search space and hence enables continuity optimisation by gradient based methods. The key of this paradigm lies in how to construct a continuous search space. For example, DARTS (Liu, Simonyan, and Yang 2019) simply relaxes the search space to be continuous by introducing a mixture of weights for all the candidate operations. NAO (Luo et al. 2018) maps a neural architecture into a continuous representation via an encoder model. Notwithstanding significantly lower search cost by further using weight sharing (Pham et al. 2018), both DARTS and NAO run the risk of being easily stuck around inferior local minimums as observed in (Sciuto et al. 2019). We consider that their limitation is commonly due to weak capability of modelling the topological knowledge of the network architecture when constructing the continuous search space. As an intrinsic property of neural network in specific and directed acyclic graph (DAG) in general, topology plays a fundamentally crucial role in the process of NAS (Figure 1).

To solve the aforementioned limitation, in this work we propose the notion of Neural Graph Embedding (NGE) for

neural architecture search. In particular, NGE elegantly enables integrating the Graph Convolutional Network (GCN) (Kipf and Welling 2017) with existing solutions, including the efficient gradient-based paradigm (Figure 1), therefore allowing for modelling the topology of the network architecture by recursive message propagation among nodes in a cell. Importantly, through dedicated neural graph embedding we obtain a *continuous* search space in a principled manner. This not only facilitates the NAS design, but also enjoys favourable search efficiency even compared with existing fast GD-based NAS methods like DARTS and NAO.

Our **contributions** are summarised below:

- We propose a novel notion of Neural Graph Embedding (NGE) for NAS, characterised by jointly modelling the graphical topology of a network architecture and performing the network search in a continuous representation space. Introducing a neural graph concept and making a principled exploitation of Graph Convolutional Network, NGE addresses the limitation of the state-of-the-art methods in mining network topology knowledge, providing a generic neural architecture representation solution specially tailored for NAS.
- We demonstrate that the proposed NGE method not only achieves highly competitive accuracy performance on CIFAR-10, CIFAR-100 and ImageNet, but also significantly reduces the architecture search process (taking only 0.1 GPU day for cell search). Moreover, we show that the neural architecture discovered on CIFAR-10 by NGE can be readily *transferred* to the more challenging semantic segmentation task. We performed this test with DeepLab-v3 on PASCAL VOC 2012 and achieved 75.96% mIOU *without* the stronger COCO pretraining, consistently outperforming the state-of-the-art network architectures.

Related Works

Existing NAS methods usually fall into three paradigms: reinforcement learning (RL) based methods, evolutionary algorithm (EA) based methods, and gradient differentiable (GD) methods. For example, the policy networks in (Zoph et al. 2018; Pham et al. 2018) guide the selection of the architecture component sequentially. Some EA-based methods (Liu et al. 2018b; Real et al. 2019) evolve a population of initialised architectures with the corresponding validation accuracies as fitness. Instead of searching in a discrete search space, DARTS (Liu, Simonyan, and Yang 2019) provides a gradient optimisation NAS framework, in which the search space is relaxed to be continuous. Several works (Liu et al. 2018a; 2018b) attempt to reduce the search cost by exploring the search space progressively. Whilst different in the specific searching algorithm, all these works commonly conduct the searching process in discrete or continuous search spaces directly *without* considering the topological information of network structures as we investigate here.

A very recent work related to ours is GHN (Zhang, Ren, and Urtasun 2019), which uses the Graph HyperNetwork (GHN) to amortise the search cost by generating network weights and predict the network performance directly. While similarly considering the topology of network architectures, our method significantly differs from GHN in three aspects: (1) In graph construction, we treat the computational node of a network as the vertex, which is more intuitive and simple than considering each operation as a graphical node in GHN. (2) Unlike GHN focusing on producing network weights, we aim at reducing the search cost by joint learning the graph representations and the connections of the topology of a network architecture, resulting in a more effective and efficient solution. (3) Our NGE is a general neural graph representational method that can be easily integrated into existing different NAS paradigms. In contrast, GHN is not a generic approach due to the specific design of the graph structure, i.e. each operation as a node in the graph.

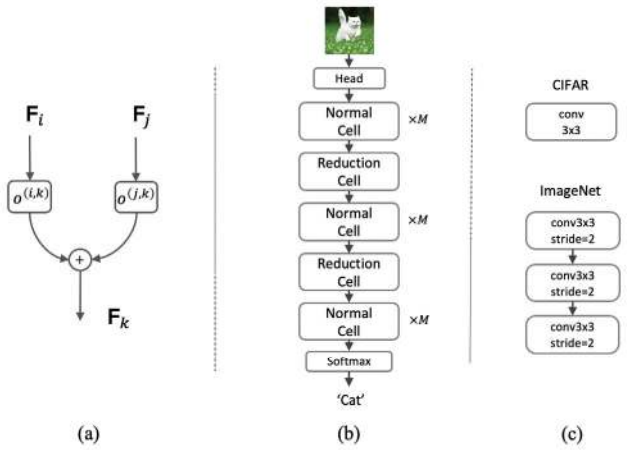

Figure 2: (a) The structure of the k-th block: taking two input feature tensors $\{\mathbf{F}_i, \mathbf{F}_j\}$, applying two separate operations $\{o^{(i,k)}, o^{(j,k)}\}$, and then combining them via element-wise addition as the output $\mathbf{F}_k$. (b) Overview of building the network by stacking $M \times 3$ normal cells and 2 reduction cells. (c) The head architectures used for CIFAR and ImageNet.

The Preliminaries

Architecture Space

Instead of an entire network architecture, a more feasible strategy is to search a repeatable structure (Zoph et al. 2018), which factorises the search space via cells and blocks.

Cell. A cell consists of a set of N ordered *feature (tensor) nodes* $\{\mathbf{F}_k|, 1 <= k <= N\}$. $\mathbf{F}_1$ & $\mathbf{F}_2$ are two *input nodes*, i.e. the outputs from the previous two cells. $\{\mathbf{F}_k\}_{k=2}^{N-1}$ denotes the *inner nodes* that perform computation. The *cell output* is the N-th node $\mathbf{F}_N$, formed as the concatenation of all the inner nodes, i.e. $\mathbf{F}_N = \texttt{concat}(\{\mathbf{F}_k\}_{k=2}^{N-1})$. There are two types of cells: *normal* cell (with stride of 1) and *reduction* cell (with stride of 2).

Block. A block is defined as a *computational node* that outputs a feature node $\mathbf{F}_k$ (Fig. 2(a)) by transforming two input feature nodes $\mathbf{F}_i$ and $\mathbf{F}_j$ as:

$$\mathbf{F}_k = o^{(i,k)}(\mathbf{F}_i) + o^{(j,k)}(\mathbf{F}_j) \quad \text{s.t.} \quad i < k \;\&\; j < k, \quad (1)$$

where $o^{(i,k)}$ and $o^{(j,k)}$ are the i-th and j-th operations. Each operation is taken from the candidate set $\mathbb{O}$ with $O = 7$ prim-

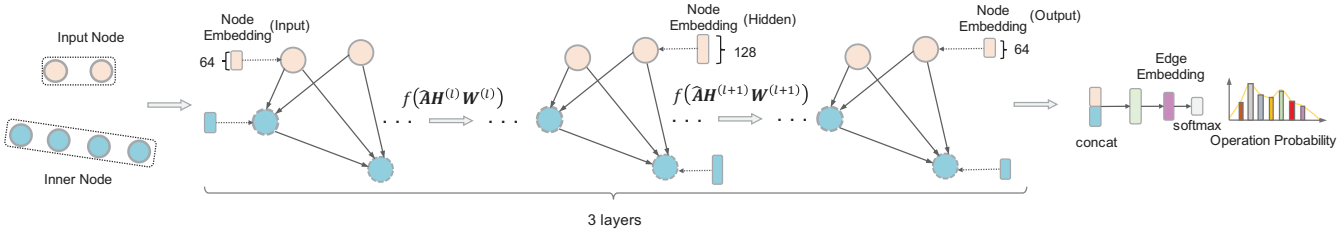

Figure 3: An overview of the proposed Neural Graph Embedding (NGE) for NAS. Each *node* denotes a computational transformation in a cell, initialised as a one-hot vector sequentially. We use a 3-layers GCN to perform the propagation of node-to-node interaction information. Each *edge* represents a connection between two nodes. We represent it by mapping the concatenation of the embeddings of the two nodes with a multilayer perceptron (MLP). It is this edge representation that significantly facilitates the optimisation of operation selection, e.g. learning an operation class classifier end-to-end.

itive operations: (1) identity, (2) 3×3 max pooling, (3) 3×3 average pooling, (4) 3×3 separable convolution, (5) 5×5 separable convolution, (6) 3×3 dilated separable convolution, (7) 5×5 dilated separable convolution.

Search Space. Generally, the architecture search space $\mathbb{A}$ is determined by the compositions of blocks, since the structure design of the input and output nodes in a cell is fixed. For a cell with $N = 7$ nodes, we only need to specify the inputs and operations for 4 *inner* computational nodes (blocks), resulting in a total number of $\prod_{n=1}^{N-3} \frac{(n+1)n}{2} \times O^2 \approx 10^9$ possible design choices.

Whole Network. Based on the definitions of cell and block above, one can construct a network in two steps: (i) Design a cell structure that contains $(N-3)$ ordered blocks; (ii) Stack multiple cells together. As shown in Fig. 2 (b), after the cell search is finished, M normal cells are stacked repeatedly for 3 times, interpolated with 2 reduction cells for downsampling the feature maps.

NAS as Optimisation

As DARTS (Liu, Simonyan, and Yang 2019), the search problem can be *efficiently* formulated in a gradient differentiable manner by relaxing the search space to be continuous.

Continuous Relaxation. For a connection from the i-th node to the k-th node in a cell with the *architecture parameters* $\boldsymbol{a}^{(i,k)}$, a softmax over all possible operations is applied to obtain the categorical choice of a particular operation:

$$\overline{o}^{(i,k)}(\mathbf{F}_i) = \sum_{o \in \mathbb{O}} \frac{\exp\left(a_o^{(i,k)}\right)}{\sum_{o' \in \mathbb{O}} \exp\left(a_{o'}^{(i,k)}\right)} o(\mathbf{F}_i). \quad (2)$$

Optimisation. Within a continuous search space, a common search process for neural architecture is generally composed of two separate optimisation procedures. Given the network parameter space $\mathbb{W}$ and the architecture space $\mathbb{A}$, the first procedure (Eq. (3)) discovers the optimal parameters $\boldsymbol{w}^* \in \mathbb{W}$ for a given architecture $\boldsymbol{a} \in \mathbb{A}$ w.r.t a training objective function $\mathcal{L}_{train}$:

$$\boldsymbol{w}^*(\boldsymbol{a}) = \arg\min_{\boldsymbol{w}} \mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{a}). \quad (3)$$

The second procedure (Eq. (4)) then explores the optimal architecture $\boldsymbol{a}^*$ over the architecture space $\mathbb{A}$ w.r.t a validation objective function $\mathcal{L}_{val}$:

$$\boldsymbol{a}^* = \arg\min_{\boldsymbol{a}} \mathcal{L}_{val}(\boldsymbol{w}^*(\boldsymbol{a}), \boldsymbol{a}). \quad (4)$$

Once this alternated optimisation is done, an amenable cell architecture is deviated by retaining the top-k strongest incoming operations from all the previous nodes.

Methodology

We aim to make full use of the intrinsic topology information of neural networks for facilitating the optimisation of NAS. To this end, we propose the notion of *Neural Graph Embedding* (NGE). The idea is that, we represent the cell and block structures with a neural graph, and leverage Graph Convolutional Networks (GCN) (Kipf and Welling 2017) to form the relational embeddings of this neural graph. Not only does our method capture the underlying topology information of network architecture comprehensively, but it also creates a means of representing the discrete operator selection by continuous feature vectors that substantially facilitate the optimisation of operator association. An overview of the proposed NGE model is depicted in Fig. 3.

In the followings, we first describe how to build a search space as a graph. We then provide the detail of GCN in the context of neural architecture graph. Finally, we delve into the details of how we integrate NGE into the task of NAS.

Neural Graph

Rather than searching over a search space $\mathbb{A}$ *directly*, we transform the architecture search space in the form of a graph. This forms a *neural graph*, leading to two advantages: (1) It explicitly encodes the high-order relationships between different nodes in a cell; (2) It also implicitly regulates the relationships between nodes and operations.

Search Space as Neural Graph. As discussed above, given the factorised search space, all we need for NAS is to search an appropriate design of blocks in a cell. Intrinsically, a cell with N ordered nodes can be defined as a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ has an associated embedding vector $\boldsymbol{x}_v \in \mathcal{X}$ (Note that the output node is excluded for consideration as its incoming connections are fixed); And the edge $e_{u \to v} = (u, v) \in \mathcal{E}$ is the connection between node u and node v, representing the information flow $u \to v$. Besides, a specific operation $o_{u,v}$ from

the candidate set $\mathbb{O}$ is applied to the edge $e_{u\to v}$. Forming this *neural graph* search space $\mathcal{G}$, next we aim to learn continuous embeddings (representations) for the nodes and the edges of $\mathcal{G}$.

Node Embedding. We learn the node embedding by designing a Graph Convolutional Network (GCN). This allows naturally modelling the topological relationships between nodes. Specifically, the input to each node v is an initial embedding vector $\boldsymbol{x}_v$, initialised as a specific one-hot vector different for each node and updated simultaneously during learning. We summarise the inputs of all the nodes as a matrix $\boldsymbol{X} = \left[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{|\mathcal{V}|}\right] \in \mathbb{R}^{|\mathcal{V}|\times D}$, where D denotes the dimension of the input embedding. The GCN outputs a node-level representation $\boldsymbol{Z} = \left[\boldsymbol{z}_1, \cdots, \boldsymbol{z}_{|\mathcal{V}|}\right] \in \mathbb{R}^{|\mathcal{V}|\times F}$, where F denotes the dimension of the output embedding.

Formally, the node embedding learning is formulated as:

$$\boldsymbol{Z} = \texttt{GCN}(\boldsymbol{X}; \boldsymbol{\Theta}_n), \tag{5}$$

where $\boldsymbol{\Theta}_n$ is the parameter for the GCN model. More specifically, considering $|\mathcal{V}|$ ordered nodes, the structure of graph search space is represented as a normalised adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ as the follows:

$$A_{i,j} = \begin{cases} \frac{1}{i+1} & \text{if } i < j \,\&\, i > 1, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

To incorporate self-reinforcement, we further form an augmented version $\hat{\boldsymbol{A}}$ by:

$$\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}, \tag{7}$$

where $\boldsymbol{I} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the identity matrix. Let $f(\cdot)$ denotes the ReLU activation function, we then formulate the per-layer learning module as:

$$\boldsymbol{H}^{(l+1)} = f(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}), \tag{8}$$

where $\boldsymbol{W}^{(l)} \in \boldsymbol{\Theta}_n$ is the l-th layer's parameter, and L the total layer number. In particular, $\boldsymbol{H}^{(0)} = \boldsymbol{X}$ and $\boldsymbol{H}^{(L)} = \boldsymbol{Z}$. With this formulation, the topological knowledge between different nodes can be continuously propagated in a stack of feed-forward operations sequentially, enabling to reveal high-order relationships across the whole neural graph.

Edge Embedding. We learn the edge embedding based on the embeddings of the two associated nodes. Specifically, for an edge between the i-th and j-th nodes, we first concatenate their embeddings $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ to merge their information. Then, we deploy an efficient MLP with two fully-connected (FC) layers and ReLU activation to further learn the edge embedding $\boldsymbol{e}^{(i,j)}$. Formally, we formulate the edge embedding learning as the follows:

$$\boldsymbol{e}^{(i,j)} = \texttt{MLP}(\texttt{concat}(\boldsymbol{z}_i, \boldsymbol{z}_j); \boldsymbol{\Theta}_e) \in \mathbb{R}^K, \tag{9}$$

where $\boldsymbol{\Theta}_e$ is the parameter set of the edge embedding MLP, shared for all the edges, and K denotes the dimension of edge embedding. Importantly, the edge embedding $\boldsymbol{e}^{(i,j)}$ not only encodes the *local* pairwise relationships between the i-th and j-th nodes, but also considers the *global* higher-order relationships among all the nodes. In doing so, we provide a principled method for modelling the comprehensive topological knowledge of a neural architecture.

Algorithm 1: Neural Graph Embedding (NGE) for NAS

Input: Training set: $\mathcal{X}_{train}$; validation set: $\mathcal{X}_{val}$.
Output: Network architecture $\boldsymbol{a}^*$

$\boldsymbol{w}$, $\boldsymbol{X}$, $\boldsymbol{\Theta}_n$, $\boldsymbol{\Theta}_e$, $\boldsymbol{\Theta}_o \leftarrow$ random initialisation
for *Num. of max epochs.* **do**
  $\boldsymbol{Z} \leftarrow$ obtain by Eq. (5);
  $\boldsymbol{E} \leftarrow$ obtain by Eq. (9);
  $\boldsymbol{P} \leftarrow$ obtain by Eq. (10);
  for *samples in* $\mathcal{X}_{train}$ **do**
    Update weights $\boldsymbol{w}$ by descending $\nabla_{\boldsymbol{w}}\mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{P})$;
  end
  for *samples in* $\mathcal{X}_{val}$ **do**
    $\boldsymbol{Z} \leftarrow$ obtain by Eq. (5);
    $\boldsymbol{E} \leftarrow$ obtain by Eq. (9);
    $\boldsymbol{P} \leftarrow$ obtain by Eq. (10);
    Update $\boldsymbol{X}$, $\boldsymbol{\Theta}_n$, $\boldsymbol{\Theta}_e$, $\boldsymbol{\Theta}_o$ by descending $\nabla\mathcal{L}_{val}(\boldsymbol{w}, \boldsymbol{P})$;
  end
end
Derive the final architecture $\boldsymbol{a}^*$ based on the learned $\boldsymbol{P}$.

NGE for NAS

Applying the NGE to the NAS task is straightforward, since an edge $e_{u\to v} = (u, v) \in \mathcal{E}$ can be readily associated with the operation selection. That being said, this allows us to derive the optimal operation selection from the edge embeddings $\boldsymbol{E}$ in a standard learning framework. It is worth mentioning that our NGE is a *general representation model* that can be integrated into different NAS paradigms. For RL-based NAS, we can compute the actions from the edge embeddings for choosing operations. For EA-based NAS, the edge embeddings act as a controller to generate mutations. In this work, due to the resource constraint we focus on the efficient GD-based NAS. Concretely, we predict the operation probability distribution for all the relaxed connections by using the NGE edge embeddings as input.

Operation Probability. Given the edge embedding $\boldsymbol{e}^{(i,j)}$ between the i-th and j-th nodes, we compute the associated operation probability $\boldsymbol{p}^{(i,j)} \in \mathbb{R}^O$ by a FC layer with the softmax activation:

$$\boldsymbol{p}^{(i,j)} = \texttt{softmax}(\texttt{FC}(\boldsymbol{e}^{(i,j)}; \boldsymbol{\Theta}_o)), \tag{10}$$

where $\boldsymbol{\Theta}_o$ is the parameter for the FC layer and shared for all the edges. We summarise the operation probability of all the edges as a matrix $\boldsymbol{P} = \left[\boldsymbol{p}_1, \cdots, \boldsymbol{p}_{|\mathcal{E}|}\right] \in \mathbb{R}^{|\mathcal{E}|\times O}$. We reformulate the continuous relaxation in Eq. (2) as:

$$\overline{o}^{(i,k)}(\mathsf{F}_i) = \sum_{o\in\mathbb{O}} p_o^{(i,k)} o(\mathsf{F}_i). \tag{11}$$

In this way, we can integrate the NGE learning into an existing GD-based NAS framework seamlessly.

Learning. In the search process, we jointly learn the NGE and the network parameters $\boldsymbol{w}$ in a fully differentiable manner. Unlike DARTS (Liu, Simonyan, and Yang 2019) optimising for each batch input, we formulate the optimisation

in an epoch-wise way, which would provide better converge speed (see a comparison in experiments). The pseudo code of NES for NAS is summarised in Algorithm 1.

Experiments

To show the effectiveness and transferability of our NGE method on both image classification and semantic segmentation tasks, we conducted the network architecture search on CIFAR-10 *only*, and compared the obtained architecture with both state-of-the-art human-design and NAS models on CIFAR-10, CIFAR-100, ImageNet and PASCAL VOC 2012 datasets. Below we gave the experiment details including datasets, model instantiating, evaluation, and analysis.

Datasets

CIFAR. Both CIFAR-10 and CIFAR-100 (Krizhevsky and others 2009) contain 50K/10K train/test RGB images with a unified resolution of 32×32. The images of both datasets are categorised into 10 and 100 classes, respectively.

ImageNet. For the large-scale image classification evaluation, we used the ILSVRC2012, a subset of ImageNet (Russakovsky et al. 2015) that contains 1K classes, 1.28M training images, and 50K validation samples.

PASCAL VOC 2012. We used the PASCAL VOC 2012 (Everingham et al. 2015) for semantic segmentation evaluation. It consists 1,464/1,449/1,456 train/val/test images with pixel annotation from 21 classes. Extra annotations from (Hariharan et al. 2011) were used for data augmentation, resulting in 10,582 training images. We used mean pixel intersection-over-union (mIOU) across all the classes to measure the performance.

NGE Instantiating

We constructed the graph search space $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = 6$ nodes (2 input nodes and 4 inner nodes). As such, there are $|\mathcal{E}| = 14$ edges totally. For the node embedding, we set the input dimension $D = 64$ and the output dimension $F = 64$. We used a ($L = 3$)-layers GCN with the hidden dimensions of 128. A MLP with 2-FC layers at the dimension of 64 was applied to learn the edge embedding with dimension $K = 64$, taking as input the concatenation of two node embeddings. We included $O = 7$ primitive operations in the candidate function set $\mathbb{O}$ as introduced early. All the parameters ($\mathbf{\Theta}_n$, $\mathbf{\Theta}_e$ and $\mathbf{\Theta}_o$) were randomly initialised in the normal distribution. All the FC layers use no bias.

Cell Search

We followed the setup of existing methods (Real et al. 2019; Liu, Simonyan, and Yang 2019; Liu et al. 2018a) to search the convolutional cells on CIFAR-10. A small proxy network consists of 8 cells was constructed for searching both the normal cell and the reduction cell. As shown in Fig. 2(b), two reduction cells are located at the $1/3$ and $2/3$ of the total depth of the network. The detailed head structure for CIFAR is depicted in Fig. 2(c), in which the number of initial channels is 16. We split 25K images from the training set for validation. We initialised the node embeddings $\boldsymbol{X}_{normal}$ and $\boldsymbol{X}_{reduction}$ for the normal cells and reduction cells, where

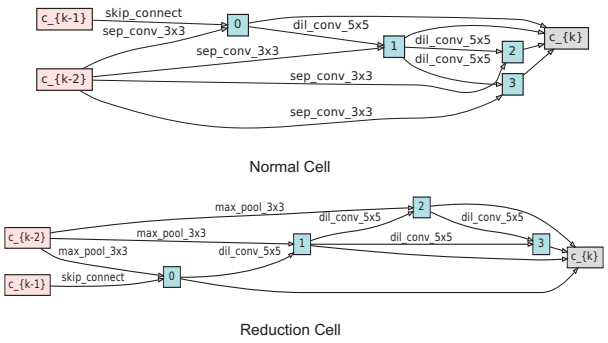

Figure 4: Normal cell and reduction cell obtained by NGE.

$\boldsymbol{X}_{normal}$ is shared for all normal cells and $\boldsymbol{X}_{reduction}$ was shared for all reduction cells. For the network parameter $\boldsymbol{w}$, we used SGD with an initial learning rate 0.025 and the momentum of 0.9. We decayed the learning rate to 0 during training using a cosine schedule. A weight decay of 3×10^{-4} was imposed to avoid over-fitting. For the NGE learning, we used the Adam optimiser with a fixed learning rate 6×10^{-4} and set the weight decay to 1×10^{-3}. To search the normal cell and reduction cell efficiently, we used 25 epochs for training the proxy network. With NGE, the search on CIFAR-10 took only 2.4 hours on a single NVIDIA Tesla V100 GPU. The searched cells by NGE is shown in Fig. 4.

Architecture Evaluation

CIFAR. To measure the final image classification performance of the searched cells on CIFAR-10 and CIFAR-100, an evaluation network of 20 cells, 36 initial channels and an auxiliary tower with loss weight 0.4 was created. The network was trained from scratch for 600 epochs with 128-sized mini-batches. To avoid over-fitting, the cutout regularisation (DeVries and Taylor 2017) with length 16 and the drop-path (Larsson, Maire, and Shakhnarovich 2017) of probability 0.3 were applied. The weight decay values for CIFAR-10 and CIFAR-100 were set to 3×10^{-4} and 5×10^{-4} individually. For model training, the standard SGD optimisation with a momentum of 0.9 was performed. The initial learning rate was 0.25, decayed to 0 with a cosine scheduler.

We summarised the evaluation results with comparison to the state-of-the-art methods in Table 1. Using NGE, the discovered network with 3.5M parameters achieves 2.60% error rate on CIFAR-10. Without re-searching, we applied the same network on CIFAR-100 and achieved 16.53% error rate. We made three observations: **(1)** NGE achieves a very competitive result (third best) on CIFAR-10, whilst enjoying the fastest search speed (only 0.1 GPU day). This demonstrates the cost-effective advantages of our NGE model, compared with ProxylessNAS (Cai, Zhu, and Han 2019) with the best accuracy and 4 GPU days and AmoebaNet-B (Real et al. 2019) with the second best accuracy and 3150 GPU days. **(2)** Compared with GHN which also conducts graph-based search, our NGE can achieve the cells with less parameters (3.4M vs 5.7M) at a significant less cost (0.1 vs

Architecture	Venue	Error (%)		Params	Search Cost		Search Method
		C10	C100	(M)	GPUs	Days	
DenseNet-BC (Huang et al. 2017)	CVPR17	3.46	17.18	25.6	-	-	manual
NASNet-A + cutout (Zoph et al. 2018)	CVPR18	2.65	-	3.3	450	1800	RL
AmoebaNet-A + cutout (Real et al. 2019)	CVPR18	3.34	-	3.2	450	3150	EA
AmoebaNet-B + cutout (Real et al. 2019)	CVPR18	2.55	-	2.8	450	3150	EA
Hireachical Evolution (Liu et al. 2018b)	ICLR18	3.75	-	15.7	200	300	EA
PNAS (Liu et al. 2018a)	ECCV18	3.41	-	3.2	100	1.5	SMBO
ENAS + cutout (Pham et al. 2018)	ICML18	2.89	-	4.6	1	0.5	RL
ProxylessNAS-G + cutout (Cai, Zhu, and Han 2019)	ICLR19	**2.08**	-	5.7	-	4	GD
RENAS (Chen et al. 2019)	CVPR19	2.88	-	3.5	4	1.5	EA&RL
DARTS (1st) + cutout (Liu, Simonyan, and Yang 2019)	ICLR19	3.00	-	3.3	1	1.5	GD
DARTS (2nd) + cutout (Liu, Simonyan, and Yang 2019)	ICLR19	2.76	17.54	3.3	1	4.0	GD
SNAS (mild) + cutout (Xie et al. 2019)	ICLR19	2.98	-	2.9	1	1.5	GD
SNAS (moderate) + cutout (Xie et al. 2019)	ICLR19	2.85	-	2.8	1	1.5	GD
SNAS (aggressive) + cutout (Xie et al. 2019)	ICLR19	3.10	-	**2.3**	1	1.5	GD
GHN + cutout (Zhang, Ren, and Urtasun 2019)	ICLR19	2.84	-	5.7	1	0.84	GD
GDAS [C=36,N=6] (Dong and Yang 2019)	CVPR19	2.93	18.38	3.4	1	0.84	GD
GDAS(FRC) [C=36,N=6] (Dong and Yang 2019)	CVPR19	2.82	18.13	2.5	1	0.68	GD
BayesNAS(0.010) + cutout (Zhou et al. 2019)	ICML19	3.02	-	2.5	1	0.2	GD
BayesNAS(0.007) + cutout (Zhou et al. 2019)	ICML19	2.90	-	3.1	1	0.2	GD
BayesNAS(0.005) + cutout (Zhou et al. 2019)	ICML19	2.81	-	3.4	1	0.2	GD
ASNG-NAS + cutout (Akimoto et al. 2019)	ICML19	2.83	-	3.9	1	0.11	GD
NGE + cutout	Ours	2.60	**16.53**	3.5	1	**0.1**	GD

Table 1: Comparisons with the state-of-the-art architectures on the CIFAR-10 and CIFAR-100 datasets.

0.84 GPU day), while obtaining a better performance (2.60 vs 2.84). **(3)** Directly transferring the CIFAR-10 searched network to CIFAR-100 can achieve the best result, outperforming DARTS (Liu, Simonyan, and Yang 2019) and GDAS (Dong and Yang 2019) significantly. This indicates the superior transferability of the network searched by our method in a challenging cross-dataset evaluation.

ImageNet. To evaluate the transferability of the architectures discovered by NGE on the large scale ImageNet benchmark, we followed the mobile setting as in (Liu, Simonyan, and Yang 2019; Dong and Yang 2019), where the number of multiply-add operations is restricted to be less than 600M with the input size at 224×224. Specifically, we constructed an evaluation network with 14 cells and 48 initial channels; The detailed head structure consists of three conv layers, as shown in Fig. 2(c). An auxiliary tower with loss weight 0.4 was also applied. We trained this model using the SGD for 250 epochs at batch-size 512 on 4 Nvidia Tesla P100 GPUs. We initialised a learning rate of 0.25 and reduced it to 0 by a linear scheduler. Learning rate warmup (Goyal et al. 2017) was applied for the first 5 epochs to deal with the large batch-size and learning rate.

The ImageNet results in the mobile setting are presented in Table 2. Notably, the cell architectures found on CIFAR-10 by our method can achieve highly competitive performance, with significantly less computation cost (0.1 GPU day *vs* 6 GPU days for RENAS and 3,150 GPU days for AmoeBaNet). Unlike ProxylessNAS searching the network on ImageNet directly using 8.3 GPU days, the network searched by NGE on CIFAR-10 can be successfully transferred. Moreover, NGE discovers the cells on CIFAR-10 that performs better on ImageNet than state-of-the-art GD-based methods (GHN, DARTS, SNAS, GDAS and BayesNAS).

Pascal VOC 2012. We further conducted a semantic segmentation experiment with DeepLabv3 (Chen et al. 2017). In this test, the Atrous Spatial Pyramid Pooling (ASPP) module, that contains three 3×3 convolutions with different atrous rates, was applied. To make a fair comparison, we followed the setting as in RENAS (Chen et al. 2019) and trained on the PASCAL VOC dataset using the above ImageNet pretrained network as the backbone model. We set the output stride to 16, which is the ratio of the input to the output spatial resolution. Note, we did *not* apply multi-scale inference and left-right flipping to improve the performance.

In Table 3, we summarise the validation set results in two pretraining settings (ImageNet and COCO (Lin et al. 2014)) and presented comparisons with other mobile networks. The results show that: **(1)** In both settings, our model achieves the best performance with 75.96% mIOU. Unlike the two state-of-the-art manually-designed models (MobileNet-v1 and MobileNet-v2), NGE does *not* rely on the stronger COCO pretraining. **(2)** Our model outperforms other two state-of-the-art NAS-designed models, whilst having less parameters (10.31M with 75.96% mIOU *vs* 12.39M with 73.68% mIOU for NASNet-A and 11.63M with 75.83% mIOU for RENAS). Overall, the cells discovered on CIFAR-10 by NGE surpass both state-of-the-art hand-crafted and NAS-mined designs on the semantic segmentation task.

Architecture	Venue	Test Err. (%)		Params	$\times+$	Search Cost	Search Method
		top-1	top-5	(M)	(M)	(GPU-days)	
MobileNet-v1(1.0)(Howard et al. 2017)	arXiv17	29.4	10.5	4.2	575	-	manual
MobileNet-v2(1.0)(Sandler et al. 2018)	CVPR18	28.0	-	3.4	300	-	manual
ShuffleNet 2× (v1) (Zhang et al. 2018)	CVPR18	26.4	10.2	≈5	524	-	manual
ShuffleNet 2× (v2) (Ma et al. 2018)	ECCV18	25.1	-	≈5	591	-	manual
NASNet-A (Zoph et al. 2018)	CVPR18	26.0	8.4	5.3	564	1800	RL
NASNet-B (Zoph et al. 2018)	CVPR18	27.2	8.7	5.3	488	1800	RL
NASNet-C (Zoph et al. 2018)	CVPR18	27.5	9.0	4.9	558	1800	RL
PNAS (Liu et al. 2018a)	ECCV18	25.8	8.1	5.1	588	1.5	SMBO
AmoebaNet-A (Real et al. 2019)	AAAI19	25.5	8.0	5.1	555	3150	EA
AmoebaNet-B (Real et al. 2019)	AAAI19	26.0	8.5	5.3	555	3150	EA
AmoebaNet-C (Real et al. 2019)	AAAI19	24.3	7.6	6.4	570	3150	EA
ProxylessNAS (GPU) (Cai, Zhu, and Han 2019)	ICLR19	24.9	7.5	7.1	**465**	8.3	GD
RENAS (Chen et al. 2019)	CVPR19	**24.3**	**7.4**	5.4	580	6	EA&RL
DARTS (Liu, Simonyan, and Yang 2019)	ICLR19	26.7	8.7	4.7	574	4.0	GD
SNAS (Xie et al. 2019)	ICLR19	27.3	9.2	4.3	522	1.5	GD
GHN (Zhang, Ren, and Urtasun 2019)	ICLR19	27.0	8.7	6.1	569	0.84	GD
GDAS [C=50,N=4] (Dong and Yang 2019)	CVPR19	26.0	8.5	5.3	581	0.84	GD
GDAS (FRC) [C=52,N=4] (Dong and Yang 2019)	CVPR19	27.5	9.1	4.4	497	0.68	GD
BayesNAS (0.010) (Zhou et al. 2019)	ICML19	28.1	9.4	4.0	-	0.2	GD
BayesNAS (0.007) (Zhou et al. 2019)	ICML19	27.3	8.4	3.3	-	0.2	GD
BayesNAS (0.005) (Zhou et al. 2019)	ICML19	26.5	8.9	3.9	-	0.2	GD
NGE (searched on CIFAR10)	Ours	25.3	7.9	5.0	563	**0.1**	GD

Table 2: Comparisons with the state-of-the-art architectures on the ImageNet benchmark with the mobile setting.

Model	Dataset	#Params	mIOU(%)
MobileNet-v1 (Howard et al. 2017)	COCO	11.15M	75.29
MobileNet-v2 (Sandler et al. 2018)	COCO	4.52M	75.70
MobileNet-v1 (Howard et al. 2017)	ImageNet	11.15M	68.79
MobileNet-v2 (Sandler et al. 2018)	ImageNet	4.52M	70.02
NASNet-A (Zoph et al. 2018)	ImageNet	12.39M	73.68
RENAS (Chen et al. 2019)	ImageNet	11.63M	75.83
NGE	ImageNet	10.31M	**75.96**

Table 3: Semantic segmentation evaluation with DeepLabv3 on the PASCAL VOC 2012 validation set.

Model	Test Error (%)	
	CIFAR-10	CIFAR-100
Baseline	3.44	17.90
Plain	3.14	17.60
RNN	2.71	16.97
NGE	**2.60**	**16.53**

Table 4: Comparing different embedding learning models.

Further Analysis

To further demonstrate the necessity of learning neural graph embeddings for NAS, we compared three alternative learning strategies: **(1)** As the *baseline* method, without neural embeddings we directly learn the architecture parameters as DARTS (Liu, Simonyan, and Yang 2019). **(2)** A *plain* embedding learning strategy is added on the baseline, in which no relationships between nodes is modelled. **(3)** Two *RNN* layers are further introduced upon the plain method to model the relationships between nodes in a sequential manner. For fair comparison, we followed the same setting as NGE to search the cell architectures on CIFAR-10 and reported the final performance on both CIFAR-10 and CIFAR-100. Table 4 shows that: (1) Compared with the baseline, learning embedding for NAS consistently helps find better cell architectures. (2) Modelling the relationships between nodes can further benefit the searching of cell architectures. (3) The proposed NGE outperforms alternative learning strategies significantly. This verifies the significance of graph embeddings for NAS and the superiority of our method.

Conclusion

We presented a generic Neural Graph Embedding (NGE) method for neural architecture search (NAS). Unlike existing methods, NGE uniquely takes into account the intrinsic topology knowledge of neural networks from the directed acyclic graph perspective. It gives rise to a generic neural network representation with the flexibility of benefiting various NAS paradigms. As an efficient showcase, we demonstrate the advantages of NGE in a gradient descent based NAS framework. Extensive experiments on image classification and semantic segmentation show that with our method, high-quality cell architectures can be identified at a significant low computation cost.

Acknowledgments

This work is supported by the China Scholarship Council, the Alan Turing Institute, and Innovate UK Industrial Challenge Project (98111-571149).

References

Akimoto, Y.; Shirakawa, S.; Yoshinari, N.; Uchida, K.; Saito, S.; and Nishida, K. 2019. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *ICML*.

Cai, H.; Chen, T.; Zhang, W.; Yu, Y.; and Wang, J. 2018a. Efficient architecture search by network transformation. In *AAAI*.

Cai, H.; Yang, J.; Zhang, W.; Han, S.; and Yu, Y. 2018b. Path-level network transformation for efficient architecture search. In *ICML*.

Cai, H.; Zhu, L.; and Han, S. 2019. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*.

Chen, L.-C.; Papandreou, G.; Schroff, F.; and Adam, H. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv*.

Chen, Y.; Meng, G.; Zhang, Q.; Xiang, S.; Huang, C.; Mu, L.; and Wang, X. 2019. Renas: Reinforced evolutionary neural architecture search. In *CVPR*, 4787–4796.

DeVries, T., and Taylor, G. W. 2017. Improved regularization of convolutional neural networks with cutout. *arXiv*.

Dong, X., and Yang, Y. 2019. Searching for a robust neural architecture in four gpu hours. In *CVPR*, 1761–1770.

Everingham, M.; Eslami, S. A.; Van Gool, L.; Williams, C. K.; Winn, J.; and Zisserman, A. 2015. The pascal visual object classes challenge: A retrospective. *IJCV* 111(1):98–136.

Ghiasi, G.; Lin, T.-Y.; and Le, Q. V. 2019. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv*.

Hariharan, B.; Arbeláez, P.; Bourdev, L.; Maji, S.; and Malik, J. 2011. Semantic contours from inverse detectors. In *ICCV*, 991–998. IEEE.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv*.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *CVPR*, 4700–4708.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

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

Larsson, G.; Maire, M.; and Shakhnarovich, G. 2017. Fractalnet: Ultra-deep neural networks without residuals.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *ECCV*, 740–755. Springer.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2018a. Progressive neural architecture search. In *ECCV*, 19–34.

Liu, H.; Simonyan, K.; Vinyals, O.; Fernando, C.; and Kavukcuoglu, K. 2018b. Hierarchical representations for efficient architecture search. In *ICLR*.

Liu, C.; Chen, L.-C.; Schroff, F.; Adam, H.; Hua, W.; Yuille, A. L.; and Fei-Fei, L. 2019. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. Darts: Differentiable architecture search. In *ICLR*.

Luo, R.; Tian, F.; Qin, T.; Chen, E.; and Liu, T.-Y. 2018. Neural architecture optimization. In *NeurIPS*, 7816–7827.

Ma, N.; Zhang, X.; Zheng, H.-T.; and Sun, J. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 116–131.

Nekrasov, V.; Chen, H.; Shen, C.; and Reid, I. 2019. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *CVPR*.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. In *ICML*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *AAAI*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *IJCV* 115(3):211–252.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 4510–4520.

Sciuto, C.; Yu, K.; Jaggi, M.; Musat, C.; and Salzmann, M. 2019. Evaluating the search phase of neural architecture search. *arXiv*.

Xie, S.; Zheng, H.; Liu, C.; and Lin, L. 2019. Snas: stochastic neural architecture search. In *ICLR*.

Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 6848–6856.

Zhang, C.; Ren, M.; and Urtasun, R. 2019. Graph hypernetworks for neural architecture search. In *ICLR*.

Zhou, H.; Yang, M.; Wang, J.; and Pan, W. 2019. Bayesnas: A bayesian approach for neural architecture search. In *ICML*.

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

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*, 8697–8710.