NASGEM: Neural Architecture Search via Graph Embedding Method

Hsin-Pai Cheng,[1] Tunhou Zhang,[1] Yixing Zhang,[1] Shiyu Li, [1] Feng Liang,[3] Feng Yan,[4] Meng Li,[2] Vikas Chandra,[2] Hai Li,[1] Yiran Chen,[1]

[1] Duke University
[2] Facebook, Inc
[3] Tsinghua University
[4] University of Nevada, Reno

{dave.cheng, tunhou.zhang, shiyu.li, hai.li, yiran.chen}@duke.edu, {meng.li, vchandra}@fb.com, liangf16@tsinghua.org.cn, fyan@unr.edu

Abstract

Neural Architecture Search (NAS) automates and prospers the design of neural networks. Estimator-based NAS has been proposed recently to model the relationship between architectures and their performance to enable scalable and flexible search. However, existing estimator-based methods encode the architecture into a latent space without considering graph similarity. Ignoring graph similarity in node-based search space may induce a large inconsistency between similar graphs and their distance in the continuous encoding space, leading to inaccurate encoding representation and/or reduced representation capacity that can yield sub-optimal search results. To preserve graph correlation information in encoding, we propose NAS-GEM which stands for Neural Architecture Search via Graph Embedding Method. NASGEM is driven by a novel graph embedding method equipped with similarity measures to capture the graph topology information. By precisely estimating the graph distance and using an auxiliary Weisfeiler-Lehman kernel to guide the encoding, NASGEM can utilize additional structural information to get more accurate graph representation to improve the search efficiency. GEMNet, a set of networks discovered by NASGEM, consistently outperforms networks crafted by existing search methods in classification tasks, i.e., with 0.4%-3.6% higher accuracy while having 11%-21% fewer Multiply-Accumulates. We further transfer GEM-Net for COCO object detection. In both one-stage and two-stage detectors, our GEMNet surpasses its manually-crafted and automatically-searched counterparts.

1 Introduction

Neural architecture search (NAS) (Zoph et al. 2018) prospers the neural architecture design process. With the rise of NAS, automated crafted CNN models have achieved record-breaking performance for a variety of vision applications such as image classification and object detection.

The goal of NAS is to identify good architectures within a designated search space under application and resource constraints. Earlier NAS works mainly adopt reinforcement learning (Zoph et al. 2018), evolutionary algorithm (Real et al. 2019), bayesian optimization (Kandasamy et al. 2018), and differentiable (Liu, Simonyan, and Yang 2019; Luo et al. 2018; Liang et al. 2019; Chen et al. 2019) methods for searching. However, these methods usually suffer from poor search scalability. Fast search methods (e.g., differentiable-based methods (Liu, Simonyan, and Yang 2019; Luo et al. 2018))

usually result in sub-optimal solutions while reward function based search methods (e.g., RL (Tan et al. 2019)) obtain high quality solutions at the cost of high computing hours. There is no effective way to trade-off search cost and architecture quality using these NAS methods.

To address the above issues, estimator-based methods (Baker et al. 2017; Li et al. 2020) are proposed to enable scalable and flexible architecture search. Estimator-based NAS formulates a representation of architecture by mapping architectures into a latent space. Such representation enables modeling the relationship between architecture and accuracy using an estimator, such as a supervised predictor. The estimator-based approach is scalable as the trade-off between search cost and quality can be controlled by budgeting the number of samples for modeling the search space. Estimator also allows adding additional search objectives with no additional search cost. To formulate an effective representation, recent works use graph convolutional networks (GCN) and other graph encoding schemes (Li, Gong, and Xiatian 2020; Ning et al. 2020) to capture the graph topology, which is important for node-based NAS.

However, existing estimator-based methods (Li, Gong, and Xiatian 2020; Ning et al. 2020; Wen et al. 2019) overlook the graph distance when mapping architectures to a latent space. This results in an inaccurate and/or reduced representation capacity of the projected latent space. When the graph distance in the latent space cannot appropriately reflect the graph distance in the discrete space, the found cell may not be optimal.

To address the above problem, we propose NASGEM (Neural Architecture Search via Graph Embedding Method) to incorporate graph kernel equipped with a similarity measure into the estimator-based search process. NASGEM delicately encodes graphs into a latent space and enables to search cells with high representation capacity. The main contributions of our work can be summarized as follows: **1)** NASGEM constructs a graphically meaningful latent space to improve the search efficiency of estimator-based method. **2)** NASGEM employs an efficiency score predictor to model the relationship between cell structures and their performances. With the pretrained graph embedding, our predictor can accurately estimate the model performance based on the representation of cell structures in a graph vector. **3)** The exploration

of optimal cell structures is further improved by bootstrap optimization, which guarantees the feasibility of graph vectors in the latent embedding space.

Our evaluation demonstrates that GEMNet outperforms models obtained by other estimator-based and node-based NAS methods on multiple vision tasks with 13%-62% parameter reduction and 11%- 21% Multiply-Accumulates (MAC) reduction. Evaluation using NASBench-101 further verifies the effectiveness of our method.

2 Related Work

Graph Embedding. Graph embedding (Goyal and Ferrara 2018; Grover and Leskovec 2016) projects graph structure into a continuous latent space. Traditional vertex graph embedding maps each node to a low-dimensional feature vector while preserving the connection relationship between vertices. Factorization-based methods (Roweis and Saul 2000), random-walk based methods (Perozzi, Al-Rfou, and Skiena 2014), and deep-learning based methods (Wang, Cui, and Zhu 2016) are popular approaches used in traditional vertex graph embedding. However, it is challenging to apply the existing graph embedding methods to deep neural networks (DNNs) for the following two reasons: (1) similarities among cell structures of DNNs cannot be explicitly derived from traditional graph embeddings; (2) sophisticated deep-learning based methods like DNGR (Cao, Lu, and Xu 2016) and GCN (Kipf and Welling 2016; Defferrard, Bresson, and Vandergheynst 2016; Li, Gong, and Xiatian 2020) require complex mechanisms when training on structural data. NASGEM addresses these issues by computing the cosine similarity of two embedded graph vectors. It also facilitates the training process during the formulation of graph embedding by employing an encoding structure.

Estimator-based NAS. Estimator-based NAS (Wen et al. 2019; Ning et al. 2020; Li et al. 2020) is mainly adopted to model the entire search space by leveraging the information of observed architectures. Estimator-based NAS is able to explore architectures within unobserved search space, which is otherwise neglected by other NAS methods. Existing estimator-based NAS works use widely adopted embedding methods such as graph convolution networks (GCN) and autoencoders (Wen et al. 2019; Li, Gong, and Xiatian 2020; Luo et al. 2018) without spending delicate efforts to improvise neural architecture representations. Specifically, these works do not consider graph distance and similarity measures while utilizing topological information in the search space. While applied to a node-based search space, these methods usually suffer from large isomorphic graph variance in embedding space and lead to the exploration of sub-optimal blocks. NASGEM uses a kernel-guided graph encoder to jointly learn graph topology and graph similarity while exploring architectures in the node-based search space. As a result, NASGEM enables a more precise prediction of the neural architecture performance and exploration of higher-quality building blocks.
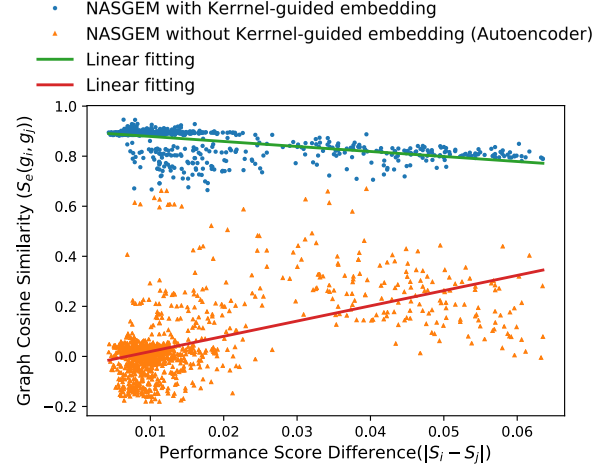

Figure 1: Each dot represents the performance score distance and the cosine similarity of vectored representation. With graph embedding (blue circle dots), similar graph pairs tends to have small performance score distance.

3 Neural Architecture Search via Graph Embedding Method

Our key intuition is that similar graphs should yield similar neural representations. For instance, given two arbitrary graphs, the graph distance between these graphs should match their difference in the representation spaces. However, the vanilla autoencoder (Poole, Sohl-Dickstein, and Ganguli 2014) is used by most estimator-based NAS methods (Luo et al. 2018; Zhang et al. 2019) and overlook such topological information. As shown in Fig. 1, the aforementioned autoencoder fails to exploit the negative correlation (orange triangle) between performance score difference and pairwise graph similarity. The incorrect correlation is due to the arbitrary representation learned by the autoencoder, therefore we envision a kernel-guided mechanism to formulate an embedding that can preserve topological information in the learned neural representation. This motivates us to develop NASGEM, a node-based neural architecture search method composed of a kernel-guided encoder to learn an effective embedding, an estimator built upon the embedding to utilize topological information, and a bootstrap optimization approach to finalize the design of high-performing neural architectures.

Fig. 2 depicts the 3-step workflow of NASGEM. In the first step, we construct a kernel-guided encoder to derive graph vectors from candidate graphs. The encoder is trained to jointly minimize reconstruction loss and pairwise graph similarity loss, see Figure 2(a). In the second step, we utilize the pretrained graph encoder to model the relationship between graph vectors and their corresponding performance. An efficiency score predictor is introduced as an estimator during the exploration process, see Figure 2(b). Finally, we obtain the optimal cell structure by applying bootstrap optimization in a large sample space. The cell structure with the highest score determined by the efficiency score predictor is adopted as the optimal building block, see Fig. 2(c).
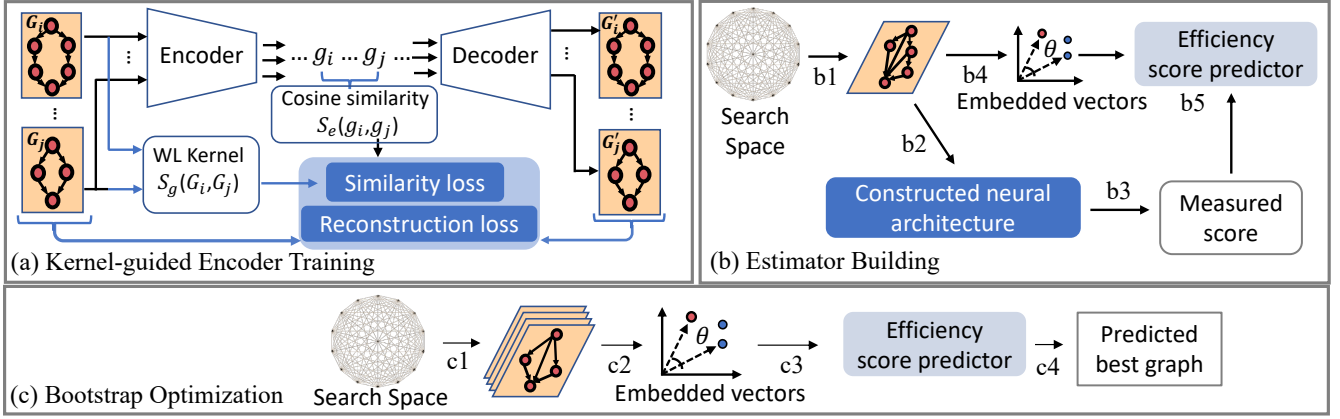
Figure 2: Workflow of NASGEM. (a) Encoder Training: the encoder learns to map graphs into a continuous embedding space by jointly minimizing the reconstruction loss and graph similarity loss. (b) Estimator Building: we *(b1)* randomly sample graphs from the search space; then *(b2)* build these graphs into neural networks and *(b3)* measure their efficiency scores; *(b4)* we also embed each sub-graph into a continuous vector with the trained kernel-guided encoder; finally, *(b5)* we train the efficiency score predictor with the embedded vectors and the corresponding efficiency score. (c) Bootstrap Optimization: we *(c1)* sample a large amount of graphs from the search space, *(c2)* embed then into vectors with the kernel-guided encoder and *(c3)* obtain the predicted efficiency score; *(c4)* we select the graph with the highest predicted score as the candidate.
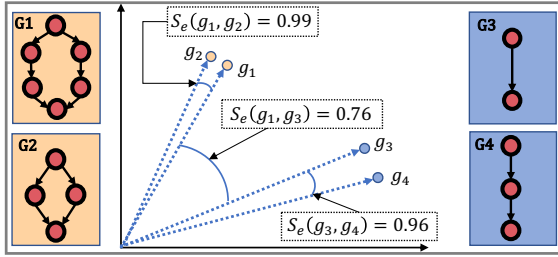

Figure 3: Our goal is to construct an embedding space such that the distance of graphs in the embedding space reflects its graph similarity. Graphs with more graph similarity (e.g., G3 and G4) have closer distance (S_g) in the embedding space.

3.1 Kernel-guided Encoder Training

In step (a) of NASGEM workflow, we train kernel-guided encoder, $\mathbf{E} : \mathbb{R}^{n \times n} \longrightarrow \mathbb{R}^d$, to vectorize the adjacency matrix of a graph to an inner product space that can represent the topological graph structure, node information (DNN operation), neighboring connections (distribution of tensors), and the pairwise graph similarity.

To measure the similarity of graphs in discrete topological space, there are various approaches in graph theory, such as WL kernel (Shervashidze et al. 2011). For continuous graph embedding space, cosine similarity is widely used to measure the similarity of graph vectors in $[-1, 1]$. The encoded vectors aim at preserving graph similarity measured by cosine similarity in the continuous space, see Fig. 3. Therefore, we train the encoder with the objective of minimizing the difference between WL kernel value in the discrete graph space and the cosine similarity value in the continuous space.

Weisfeiler-Lehman (WL) Kernel. To estimate graph similarity, a common method is to learn a positive definite kernel, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ represents the adjacency matrix

set. Here we adopt Weisfeiler-Lehman (WL) kernel (Shervashidze et al. 2011) as a graph similarity measure of two arbitrary input graphs $\mathbf{G}_i, \mathbf{G}_j$ as follows:

$$S_g(\mathbf{G}_i, \mathbf{G}_j) = k_{WL}^{(h)}(\mathbf{A}_i, \mathbf{A}_j), \qquad (1)$$

where $\mathbf{A}_i, \mathbf{A}_j$ are the adjacency matrices for graph $\mathbf{G}_i, \mathbf{G}_j$. h denotes the iteration times of computing WL kernel. For graphs with N nodes, the WL kernel can be computed with h iterations in $O(h \times N)$. Compared with other graph similarity metrics, WL kernel is able to measure large and complex graphs with relatively low computational complexity. The procedure and code of measuring the similarity between two input graphs is provided in the supplementary material.

Similarity measure of graph vectors. We adopt cosine similarity to measure the similarity of the vectorized graph representations (i.e., graph vectors). The cosine similarity measure on the embedding space is defined as:

$$S_e(\mathbf{g}_i, \mathbf{g}_j) = \frac{\mathbf{g}_i \cdot \mathbf{g}_j}{||\mathbf{g}_i|| \, ||\mathbf{g}_j||}, \qquad (2)$$

where $\mathbf{g}_i$ and $\mathbf{g}_j$ are vectorized representation of graph $\mathbf{G}_i$ and $\mathbf{G}_j$. The direction of the vector reflects the proximities of the original graphs. Compared with the standard Euclidean distance in $\mathbb{R}^n$, cosine similarity is scale-invariant. In addition, since the range of cosine distance is bounded, it is comparable with the similarity value given by WL kernel. Thus we can supervise the training of the embedding function with the difference between the cosine similarity value and the WL kernel value.

Encoder training objective. As graph similarities are always measured pairwise, the encoder is trained and evaluated on a large number of graph pairs $(\mathbf{G}_i, \mathbf{G}_j)$. To formulate a mapping to represent topological graph structure in a continuous space, the cosine similarity of the encoded graph vectors shall represent the similarity of the corresponding original

graph in the original discrete topological space. To satisfy the above training objectives, we define the similarity loss with respect to a pair of input graph: $(\mathbf{G}_i, \mathbf{G}_j)$ as:

$$\mathcal{L}_s(\mathbf{G}_i, \mathbf{G}_j; \mathbf{E}) = [S_e(\mathbf{E}(\mathbf{A}_i), \mathbf{E}(\mathbf{A}_j)) - S_g(\mathbf{G}_i, \mathbf{G}_j)]^2. \quad (3)$$

Besides similarity loss, we also consider reconstruction loss, $\mathcal{L}_r(\mathbf{G}_i, \mathbf{D}(\mathbf{E}(\mathbf{G}_i)))$, for each graph. Here $\mathbf{D}$ is a decoder that maps the encoded graph to the same dimension as $\mathbf{G_i}$ Therefore, our final loss function of encoder training is as follow:

$$\mathbf{E}^* = \min_{\mathbf{E}} \sum_{i,j} \{ \mathcal{L}_s(\mathbf{G}_i, \mathbf{G}_j; \mathbf{E}) \\ + \mathcal{L}_r(\mathbf{G}_i, \mathbf{D}(\mathbf{E}(\mathbf{G}_i))) + \mathcal{L}_r(\mathbf{G}_j, \mathbf{D}(\mathbf{E}(\mathbf{G}_j))) \}. \quad (4)$$

Following common practice in autoencoders (Hinton and Zemel 1994), we adopt fully-connected feedforward networks for both encoder and decoder. The encoder is trained independently before the searching procedure. We randomly generate a number of graph structure pairs with the same numbers of nodes and apply WL kernel to provide ground truth labels. Details and code implementation can be found in supplementary material.

3.2 Estimator Building

Performance estimation of graph vectors. Like most of the previous works (Liu, Simonyan, and Yang 2019; Pham et al. 2018), we map DAGs to cell structures, which are used as building blocks of DNNs. Each node in the DAG stands for a valid DNN operation (such as depth-wise separable convolution 3×3), and each edge represents the flow of tensors from one node to another. When mapping cell structures to DNN architectures, nodes with no input connections (i.e., zero in-degree) are dropped while nodes with in-degree larger than one is inserted a concatenate operation. The output of building blocks can be constructed from the leaf nodes with zero out-degree. The concatenation of these leaf nodes along the last dimension gives the output feature maps.

To evaluate the performance of DNN architectures formed by the corresponding cell structures of graph vectors, we use efficiency score instead of accuracy as our search metric so that both performance and efficiency are taken into considerations. The efficiency score for candidate graph $\mathbf{G}$ is formulated as:

$$S(\mathbf{G}) = ACC[N(\mathbf{G})] - \lambda \log(MAC[N(\mathbf{G})]), \quad (5)$$

where $N(\mathbf{G})$ is the DNN constructed with the cell represented by $\mathbf{G}$. ACC is the validation accuracy on proxy dataset. MAC is the number of Multiply-Add operations of $N(\mathbf{G})$ measured in Millions. λ is a penalty coefficient. We use MAC as the penalty term since it can be precisely measured across search iterations. Most compact models (Howard et al. 2017; Sandler et al. 2018; Tan et al. 2019) have hundred millions of MACs while large models (Szegedy et al. 2015) can have billions of MACs. This penalty function can urge the search iteration towards improving the performance of small models or deflating the complexity of large models, and thus strike a balance between complexity and performance.

Train Efficiency Score Predictor. The efficiency score predictor $\mathbf{P} : \mathbb{R}^d \longrightarrow \mathbb{R}$ maps the d-dimensional graph vector to a real-value that indicates the performance of architectures built upon this graph vector. The predictor is a fully connected neural network with activation function ReLU. We maintain a set $\{(g, y)\}$, where g is a graph vector and y is its efficiency score measured on proxy dataset. In each iteration, we add current selected graph vector/score pair into the set and train the predictor P with this enlarged set. The predictor becomes more accurate by using the efficiency score of new samples for fine-tuning. More importantly, the predictor in NASGEM is built on top a smoother latent space, which is constructed through the graph embedding of unrestricted DAGs. Such accurate prediction allows to explore a wider search space with extremely small search cost (0.4 GPU days). Predictor training algorithm and code implementation can be found in Supplementary Material.

Fig. 1 shows the feasibility of the proposed predictor. We can see the cosine similarity of the output vectors of graph encoder is inversely proportional to their performance distance. This indicates that after training, kernel-guided encoder can learn to encode similarity information between two graphs into the intersection angle of their vectorized representations. In other words, graph embedding can improve the accuracy of predictor by enhancing robustness under isomorphic graphs.

We also show that for isomorphic or similar graphs, our predictor gives close performance score. For a fully connect neural network, its Lipschitz constant always exists (Jordan and Dimakis 2020; Virmaux and Scaman 2018). Thus, we assume the Lipschitz constant of predictor $\mathbf{P}$ after training is K, which means $\forall x_1, x_2 \in \mathbb{R}^d$,

$$|\mathbf{P}(x_1) - \mathbf{P}(x_2)| \le K\|x_1 - x_2\| \quad (6)$$

where $\|\cdot\|$ represents L^2 norm. Let the input of predictor be a random vector $X \sim \mu \in \mathbb{R}^d$. X' and X are independently drawn from probability measure μ. We have

$$\mathbb{E}_{\mu \times \mu} \Big[|\mathbf{P}(X) - \mathbf{P}(X')|^2 \Big] \quad (7)$$

$$= \mathbb{E}_{\mu \times \mu} \big[\mathbf{P}^2(X) + \mathbf{P}^2(X') - 2\mathbf{P}(X)\mathbf{P}(X') \big] \quad (8)$$

$$= 2\mathbb{E} \big[\mathbf{P}^2(X) \big] - 2(\mathbb{E}[\mathbf{P}(X)])^2 \quad (9)$$

$$= 2\mathbb{E} \Big[(\mathbf{P}(X) - \mathbb{E}[\mathbf{P}(X)])^2 \Big]. \quad (10)$$

Based on (6),

$$\mathbb{E}_{\mu \times \mu} \Big[|\mathbf{P}(X) - \mathbf{P}(X')|^2 \Big] \le K^2 \mathbb{E}_{\mu \times \mu} \Big[\|X - X'\|^2 \Big] \quad (11)$$

$$= 2K^2 \mathbb{E} \Big[(X - \mathbb{E}[X])^\top (X - \mathbb{E}[X]) \Big]. \quad (12)$$

Based on (7)-(10) and (11)-(12),

$$\mathbb{E} \Big[(\mathbf{P}(X) - \mathbb{E}[\mathbf{P}(X)])^2 \Big] \le K^2 \mathbb{E} \Big[\|X - \mathbb{E}[X]\|^2 \Big]. \quad (13)$$

Equation (13) shows the variance of the output (performance score) of predictor is upper bounded by its Lipschitz constant and the variance of input vectors. After training, the predictor is determined with a fixed Lipschitz constant K. Encoder with WL kernel embedding decreases the variance of input for isomorphic or similar graphs. Therefore, it enhances the robustness of predictor under isomorphic or similar graphs.

3.3 Bootstrap Optimization

After the predictor is trained using a large number of neural architectures and their corresponding efficiency scores, our goal of finding the optimal cell structure in the topological graph space is equivalent to finding the graph vector that has the highest score according to the efficiency score predictor, formulated as:

$$g^* = \arg\max_g \mathbf{P}(g), \tag{14}$$

where $g = \mathbf{E}^*(\mathbf{A})$ is the embedded graph vector after passing adjacency matrix $\mathbf{A}$ into the pretrained graph encoder, and $\mathbf{P}(g) : \mathbb{R}^d \longrightarrow \mathbb{R}$ is the efficiency score predictor that estimates the efficiency score of a given cell A.

NASGEM explores a continuous immense search space consisting of hyper-complex families of cell structures. For efficient exploration, we introduce an exploration method based on two empirical beliefs: (1) Optimal cell structures within the search space is not unique as various architectures of the similar isomorphism can yield equally competitive results. (2) Finding the optimal graph vector in the continuous search space and then decoding may not discover a valid architecture, as the mapping from graph vectors to discrete topological cell structures may not be injective.

For simplicity, we use *Bootstrap Optimization* to address the above issues by sampling the cell structures with replacement among a large sample space $\mathbf{S}$ and picking up the best one as our post-searching approximation. With the pretrained graph encoder $\mathbf{E}$, we randomly sample cell structure $A \in \mathbf{S}$ from the sample space and approximate the best candidate cell structure A^* by predicting the efficiency score using the efficiency score predictor $\mathbf{P}$:

$$A^* = \arg\max_{A \in \mathbf{S}} \mathbf{P}(\mathbf{E}(A)). \tag{15}$$

4 Experimental Evaluation

We apply NASGEM to search efficient mobile neural architectures on image classification and object detection tasks. We initiate architecture search with a complete DAG of $N = 30$ so that a rich source of cells can be constructed. Each node can choose from either $1{\times}1$ convolution or $3{\times}3$ depthwise separable convolution as its candidate operation. Following the common practice in NAS, each convolution operation adopts a Convolution-BatchNorm-ReLU triplet (Xie et al. 2019a; Liu, Simonyan, and Yang 2019). We use $1/10$ of the CIFAR-10 dataset as a proxy to evaluate performance, and use the obtained performance to train the predictor.

4.1 Learning Dynamics of NASGEM

We illustrate the learning dynamics of NASGEM by plotting the efficiency score surface with respect to latent vectors in the graph embedding space. The efficiency score surface is obtained by sampling 50,000 adjacency matrices, mapping them to the continuous embedding space, and passing them through the efficiency score predictor.

Fig. 4 illustrates the efficiency score surface of efficiency score predictors with and without using graph embedding as a latent vector. This can be interpreted as the embedding of the optimal cell structure in the continuous embedding space.

Under the kernel-guided embedding, the efficiency score surface is smoother, which achieves global optimum more easily and more efficiently. In contrast, without graph kernel guided embedding, the efficiency score surface cannot be constructed smoothly, which makes the optimization process more difficult and less efficient.

4.2 Classification on ImageNet

Table 1 summarizes the key performance metrics on the ImageNet dataset. For a fair comparison, we compare with the most relevant NAS works that also use either encoding scheme or node-based search space (without predefined chain-liked backbone). Note that we report the total search cost for all phases: encoder training, estimator building, and bootstrap optimization. GEMNet, the optimal architectures crafted by our explored building block, are consistently more accurate with fewer parameters and MACs. Specifically, GEMNet-A and GEMNet-B surpass all the node-based works (DARTS, P-DARTS, PC-DARTS, SNAS, GDAS, MiLeNAS). GEMNet with kernel-guided graph embedding also outperforms the most recent NAS methods with alternative embedding methods such as LSTM, GCN, and GATES.

4.3 Object Detection on COCO

To further evaluate the transferability of GEMNet, we conduct object detection experiments on the challenging MS COCO dataset (Lin et al. 2014). We use the whole COCO *trainval135* as training set and validate on COCO *minival*. For both two-stage Faster RCNN detector with Feature Pyramid Networks (FPN) (Ren et al. 2015; Lin et al. 2017a) and one-stage RetinaNet (Lin et al. 2017b) detector, the input images are resized to a short side of 800 pixels and a long side not exceeding 1333 pixels. As shown in Table 2, compared with the manually crafted MobileNetV2 (Sandler et al. 2018) and automatically searched MnasNet (Tan et al. 2019), GEMNet detection model achieves up to 0.7% higher AP with fewer parameters and lower MACs.

4.4 Evaluation on NASBench-101

To facilitate the reproducibility of NAS and evaluate search strategy, NASBench-101 (Ying et al. 2019) provides abundant results for various neural architectures (about 423K) in a large number of search spaces. To justify the effectiveness of NASGEM, we further perform evaluation on NASBench-101. We randomly sample a fixed number of candidate architectures (200∼2000) from NASBench-101 within a fixed operation list given by NASBench-101. Then we train the efficiency score predictor with/without our proposed graph embedding on these sampled architecture respectively. Finally, we select the best architecture through bootstrap optimization by using the trained efficiency score predictor to evaluate a total of 50K architectures and pick the best one.

We measure the performance of NASGEM on NASBench-101 by *Global Prediction Bias*, $B = A_B - A_P$. Here A_B is the *global accuracy*, which is the best accuracy in the whole search space given in NASBench-101. A_P is the *predicted accuracy*, which is the accuracy achieved by NASGEM. When *predicted accuracy* is very close to *global accuracy*, *global*

(a) With graph kernel guided embedding. (b) Without graph kernel guided embedding.
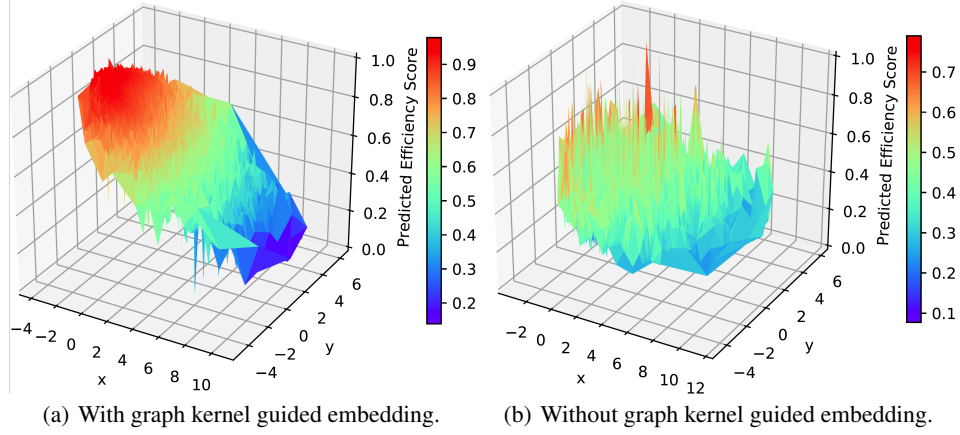
Figure 4: Efficiency score surface of efficiency score predictor based on DNN performance on CIFAR-10 dataset and computation cost in terms of MACs. We use PCA to project the graph vectors into a 2-dimensional space named x and y and plot the efficiency score of graph vector using a heatmap for (a) with graph kernel guided embedding and (b) the case of without embedding.

Architecture	Encoder	Test Err.(%)		#Params (M)	MACs (M)	Search Cost (GPU days)
		top-1	top-5			
RandWire-WS (Xie et al. 2019a)	/	$25.3_{\pm0.25}$	$7.8_{\pm0.15}$	$5.6_{\pm0.1}$	$583_{\pm6.2}$	/
DARTS (Liu, Simonyan, and Yang 2019)	/	26.7	8.7	4.7	574	4.0
P-DARTS (Chen et al. 2019)	/	24.7	7.5	5.1	577	0.3
PC-DARTS (Xu et al. 2019)	/	25.1	7.8	5.3	586	0.1
SNAS (Xie et al. 2019b)	/	27.3	9.2	4.3	522	1.5
GDAS (Dong and Yang 2019)	/	26.0	8.5	5.3	581	0.21
MiLeNAS (He et al. 2020)	/	25.4	7.9	4.9	570	0.3
BayesNAS (Zhou et al. 2019)	/	26.5	8.9	3.9	/	0.2
NAONet (Luo et al. 2018)	LSTM	25.7	8.2	11.35	584	200
NGE (Li, Gong, and Xiatian 2020)	GCN	25.3	7.9	5.0	563	0.1
GATES (Ning et al. 2020)	GATES	24.1	/	5.6	/	/
GEMNet-A (Ours)	Kernel-guided MLP	**23.7**	**7.9**	**4.3**	**463**	**0.4**
GEMNet-B (Ours)	Kernel-guided MLP	**23.3**	**7.8**	**4.5**	**563**	**0.4**

Table 1: ImageNet results with different computation budget. For fair comparison, the input image resolution is fixed at 224×224. Note that the pareto frontier of MACs and parameter count (#Params) is not linear. Further reducing computation cost under a small computation regime is very challenging as redundancy is already low.

prediction bias (B) is moving toward zero. Therefore, the smaller B reflects the more accurate prediction.

As shown in Fig. 5, with the guidance of graph kernel embedding, there is around 0.2% gain on predicting global accuracy on the NASBench-101 dataset. Such topological information can facilitate the training of efficiency score predictor given insufficient data. NASGEM also produces more stable results as structural knowledge represented by graph embedding generalizes better than binary adjacency matrices. Thus, the performance of NASGEM is less sensitive to the number of evaluated samples in the search space.

4.5 Ablation Studies

Impact of embedding dimension and number of nodes. Here we conduct an analysis on two factors of the graph encoder: number of nodes n and embedding dimension d. We choose n in [10, 30, 50, 100, 150, 200, 250], d in [10, 20, 30,
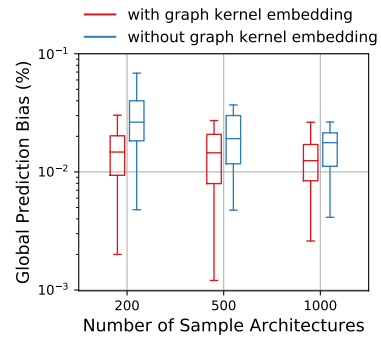

Figure 5: Efficiency score predictor's performance on NASBench-101 with and without graph kernel embedding.

40, 50], the encoder and decoder are trained for 10k iterations using 50k generated graph pairs. As shown in Figure 6, the similarity loss falls fast until convergence. Meanwhile, larger

Backbone	Detector	Params(M)	MACs(G)	AP_s	AP_m	AP_l	AP_{50}	AP_{75}	AP
MobileNetV2×1.4		14.2	170.5	18.7	36.6	44.5	52.9	35.7	33.5
MnasNet×1.3	RetinaNet	14.8	169.2	19.7	38.2	47.6	55.2	36.9	35.0
GEMNet-A (Ours)		**13.5**	**166.9**	**21.2**	38.9	47.3	55.6	37.5	35.4
GEMNet-B (Ours)		14.2	169.2	21.1	**39.3**	**47.9**	**55.8**	**38.2**	**35.7**
MobileNetV2×1.4		22.1	133.4	19.4	36.5	43.6	55.6	35.5	33.6
MnasNet×1.3	Faster RCNN	22.6	**132.1**	21.4	38.6	45.9	57.9	37.5	35.4
GEMNet-A (Ours)		**21.4**	133.1	21.7	38.7	45.1	57.6	37.8	35.3
GEMNet-B (Ours)		22.0	133.6	**21.9**	**39.0**	45.9	**58.2**	**38.1**	**35.7**

Table 2: Results on MS COCO dataset. Parameters and MACs are measured on the whole detector with input size 800×1333.

n and smaller d will introduce higher loss. It reveals that our encoder can be applied to search different sizes of graphs by adjusting embedding dimensions.
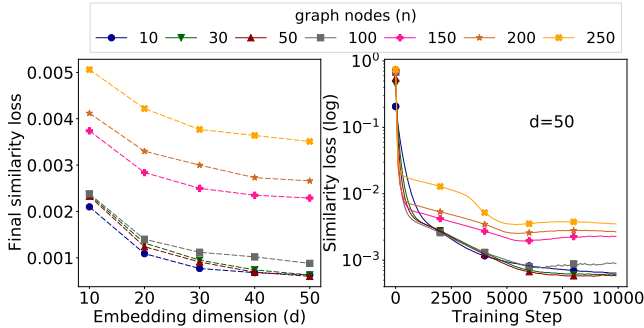

Figure 6: Kernel-guided encoder training under different number of nodes and embedding dimension.

Impact of proportion of training data. In Table 3 and 4, we show the effectiveness of the proposed embedding scheme by evaluating the impact of different embedding methods with Kendall's tau (Sen 1968) and Pearson correlation coefficient (Benesty et al. 2009).

We build NASGEM-Bench containing 1000 neural architectures to evaluate the effectiveness of our graph embedding method. To minimize the variance from the choice of node operation, each cell in NASGEM-Bench has at most 6 nodes and each node has only one kind of operation, 3×3 convolution. We randomly sample 1000 graphs to build a 3 stage DNN. We train each network on CIFAR-10 with 30 epochs and record the final accuracy.

We use 60% data (600 graphs) from NASGEM-Bench as the training samples and the rest 40% (400 graphs) as the test set. Under the 60% of training data, we randomly slice different proportions of training data for comparison. We try three methods to train our predictor based on different proportions of training data. In the first method, we directly flatten the upper triangle of the adjacency matrices, and use the flatten vector and the corresponding performance scores to train the predictor. In the second method, we first use a two-layer MLP to build an autoencoder, then use MLP encoder to embed the adjacency matrices into vectors. We use those vectors and their performance scores to train a predictor. In the third method, we use our kernel-guided encoder to embed the adjacency matrices into vectors, then train the predictor with the scores and embedding vectors. Compared with purely using an adjacency matrix and MLP-based autoencoder, NASGEM's embedding method consistently achieves better prediction under different proportion of training data.

Table 3: The Kendall's Tau of the predicted result with different embedding methods.

Embedding Method	Proportion of Training Data (%)					
	10	20	30	50	70	100
adj. matrix	0.42	0.45	0.44	0.44	0.46	0.46
MLP	0.45	0.47	0.44	0.45	0.46	0.47
NASGEM	**0.48**	**0.53**	**0.53**	**0.55**	**0.54**	**0.55**

Table 4: The Pearson correlation coefficient of the predicted result with different embedding methods.

Embedding Method	Proportion of Training Data (%)					
	10	20	30	50	70	100
adj. matrix	0.47	0.64	0.62	0.66	0.67	0.66
MLP	0.57	0.58	0.66	0.63	0.62	0.71
NASGEM	**0.76**	**0.82**	**0.84**	**0.85**	**0.85**	**0.85**

5 Conclusion

NASGEM is the first of its kind estimator-based NAS method that tackles down the limitations of graph topology exploration in existing search methods via: (i) construct a topologically meaningful representation by WL kernel guided *graph embedding*; (ii) employ an *efficiency score predictor* to precisely model the relationship between neural architectures and performance; (iii) use *bootstrap optimization* to explore the optimal architecture. All these components work coherently to enable NASGEM to search a more efficient neural architecture from an unrestricted wide search space within a short time. Compared with neural architectures produced by existing embedding methods, GEMNet crafted by NASGEM consistently achieves higher accuracy on image classification and object detection while having less parameters and MACs. NASGEM is highly adaptable to different search spaces. By combining our proposed graph embedding with NASBench-101, it achieves a more precise and stable prediction compared to the version without graph embedding.

References

Baker, B.; Gupta, O.; Raskar, R.; and Naik, N. 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823* .

Benesty, J.; Chen, J.; Huang, Y.; and Cohen, I. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*, 1–4. Springer.

Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Chen, X.; Xie, L.; Wu, J.; and Tian, Q. 2019. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, 1294–1303.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*, 3844–3852. Curran Associates, Inc. URL http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering.pdf.

Dong, X.; and Yang, Y. 2019. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1761–1770.

Goyal, P.; and Ferrara, E. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151: 78–94.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864. ACM.

He, C.; Ye, H.; Shen, L.; and Zhang, T. 2020. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Hinton, G. E.; and Zemel, R. S. 1994. Autoencoders, minimum description length and Helmholtz free energy. In *Advances in neural information processing systems*, 3–10.

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. In *arXiv preprint arXiv:1704.04861*.

Jordan, M.; and Dimakis, A. G. 2020. Exactly Computing the Local Lipschitz Constant of ReLU Networks. *arXiv preprint arXiv:2003.01219* .

Kandasamy, K.; Neiswanger, W.; Schneider, J.; Poczos, B.; and Xing, E. P. 2018. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, 2016–2025.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations*.

Li, W.; Gong, S.; and Xiatian, Z. 2020. Neural Graph Embedding for Neural Architecture Search. In *The AAAI Conference on Artificial Intellegence*.

Li, Z.; Xi, T.; Deng, J.; Zhang, G.; Wen, S.; and He, R. 2020. GP-NAS: Gaussian Process Based Neural Architecture Search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liang, H.; Zhang, S.; Sun, J.; He, X.; Huang, W.; Zhuang, K.; and Li, Z. 2019. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035* .

Lin, T.-Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; and Belongie, S. 2017a. Feature pyramid networks for object detection. In *CVPR*.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2017b. Focal loss for dense object detection. In *ICCV*.

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 *European conference on computer vision*, 740–755. Springer.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*.

Luo, R.; Tian, F.; Qin, T.; Chen, E.; and Liu, T.-Y. 2018. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, 7827–7838.

Ning, X.; Zheng, Y.; Zhao, T.; Wang, Y.; and Yang, H. 2020. A Generic Graph-based Neural Architecture Encoding Scheme for Predictor-based NAS. *arXiv preprint arXiv:2004.01899* .

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, 701–710. New York, NY, USA: ACM. ISBN 978-1-4503-2956-9. doi:10.1145/2623330.2623732. URL http://doi.acm.org/10.1145/2623330.2623732.

Pham, H.; Guan, M.; Zoph, B.; Le, Q.; and Dean, J. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *International Conference on Machine Learning*, 4092–4101.

Poole, B.; Sohl-Dickstein, J.; and Ganguli, S. 2014. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831* .

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4780–4789.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NeurIPS*.

Roweis, S. T.; and Saul, L. K. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290(5500): 2323–2326.

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

Sen, P. K. 1968. Estimates of the regression coefficient based on Kendall's tau. *Journal of the American statistical association* 63(324): 1379–1389.

Shervashidze, N.; Schweitzer, P.; Leeuwen, E. J. v.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12(Sep): 2539–2561.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1–9.

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

Virmaux, A.; and Scaman, K. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems*, 3835–3844.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 1225–1234. ACM.

Wen, W.; Liu, H.; Li, H.; Chen, Y.; Bender, G.; and Kindermans, P.-J. 2019. Neural predictor for neural architecture search. *arXiv preprint arXiv:1912.00848* .

Xie, S.; Kirillov, A.; Girshick, R.; and He, K. 2019a. Exploring Randomly Wired Neural Networks for Image Recognition. In *The IEEE International Conference on Computer Vision (ICCV)*.

Xie, S.; Zheng, H.; Liu, C.; and Lin, L. 2019b. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=rylqooRqK7.

Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.-J.; Tian, Q.; and Xiong, H. 2019. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *International Conference on Learning Representations*.

Ying, C.; Klein, A.; Christiansen, E.; Real, E.; Murphy, K.; and Hutter, F. 2019. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 7105–7114. Long Beach, California, USA: PMLR. URL http://proceedings.mlr.press/v97/ying19a.html.

Zhang, M.; Jiang, S.; Cui, Z.; Garnett, R.; and Chen, Y. 2019. D-vae: A variational autoencoder for directed acyclic graphs. In *Advances in Neural Information Processing Systems*, 1588–1600.

Zhou, H.; Yang, M.; Wang, J.; and Pan, W. 2019. BayesNAS: A Bayesian Approach for Neural Architecture Search. In *ICML*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8697–8710.