Deep Neural Architecture Search with Deep Graph Bayesian Optimization

Lizheng Ma
Jilin University
Chaoyang Qu, Changchun Shi, China

Jiaxu Cui
Jilin University
Chaoyang Qu, Changchun Shi, China

Bo Yang*
Jilin University
Chaoyang Qu, Changchun Shi, China

ABSTRACT

Image recognition aims to identify objects, places, people, or other targeted items in a given image, and has a wide range of social applications such as natural disasters recognition, plant disease detection, and traffic jam detection. Currently state-of-the-art methods of image recognition are based on deep learning and remain a common pattern in designing and using convolutional neural networks (CNNs). However, designing CNNs is extremely time intensive and requires an expert. Neural architecture search (NAS) can solve this problem by automatically identifying architectures of CNNs that are superior to hand-designed ones. Recently BO has been applied to neural architecture search and shows better performance than pure evolutionary strategies. All these methods adopt Gaussian processes (GPs) as surrogate function, with the handcraft similarity metrics as input. In this work, we propose a Bayesian graph neural network as a new surrogate, which can automatically extract features from deep neural architectures, and use such learned features to fit and characterize black-box objectives and their uncertainty. Based on the new surrogate, we then develop a graph Bayesian optimization framework to address the challenging task of deep neural architecture search. Experiment results show our method significantly outperforms the comparative methods on benchmark tasks.

CCS CONCEPTS

• **Networks** → **Network performance evaluation**; • **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

deep learning, neural architecture search, bayesian optimization, graph neural network, image recognition

ACM Reference Format:
Lizheng Ma, Jiaxu Cui, and Bo Yang. 2019. Deep Neural Architecture Search with Deep Graph Bayesian Optimization. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI '19), October 14–17, 2019, Thessaloniki, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3350546.3360740

*Corresponding author: ybo@jlu.edu.cn

WI '19, October 14–17, 2019, Thessaloniki, Greece

ACM ISBN 978-1-4503-6934-3/19/10…$15.00
https://doi.org/10.1145/3350546.3360740

1 INTRODUCTION

In recent years, image recognition[9, 15, 16] is getting more and more attention from researchers due to its ability to handle many public problems. Image recognition can make meaningful judgments based on observed images. For example, natural disasters always affect the development of human society and economic progress. Image recognition can identify and tag the type of natural disasters in a timely way by using the satellite images, and thus facilitates the corresponding protection and treatment. Disease on plant leads to the significant reduction in both the quality and quantity of agricultural products, farmers can visually monitor the situation of plant for successful cultivation of crops. With the increase in vehicle traffic, traffic jam seriously affects public daily life, and limits the economical development of society. Image recognition technologies can help human beings detect road congestion in real time by calculating traffic density and drivers can make an appropriate choice based on this information.

Convolutional neural networks offer the promise of avoiding manual feature engineering by learning representations in an end-to-end manner. The key to image recognition is how to design convolutional neural networks. To achieve good performance, these deep architectures need to be carefully designed by human experts. Due to the huge search space, it definitely takes great efforts to do this hard job, which limits use of image recognition to a smaller community of scientists and engineers. Neural architecture search (NAS), as a part of automated machine learning (AutoML), can provide researchers more effective way to design the architecture of image recognition and save researchers a lot of time, who are from academia and industry in the areas of artificial intelligence, machine learning and so on. Recently, there has been an increase of the literatures about neural architecture search [9, 12, 14]. Roughly, such works can be categorized into two main streams, one is based on reinforcement learning (RL) [19], and the other is based on evolutionary algorithm (EA) [14, 16]. EA-based methods can continually update the structures of neural nets to generate a series of generations for expanding search space through simpler and more efficient operations, and show promising results compared with other methods including RL-based ones. However, EA-based methods need to evaluate a large number of individual networks, which consumes a lot of expensive GPU resources.

To solve this problem, one can use Bayesian optimization strategy. The conventional solution of automatic machine learning resorts to formalizing machine learning process as a black-box optimization task. Bayesian optimization (BO) is an effective global optimization algorithm, with the goal of finding the optima of black-box objectives. Since the information about objective function is not known, BO utilizes a surrogate model to fit the black-box function,

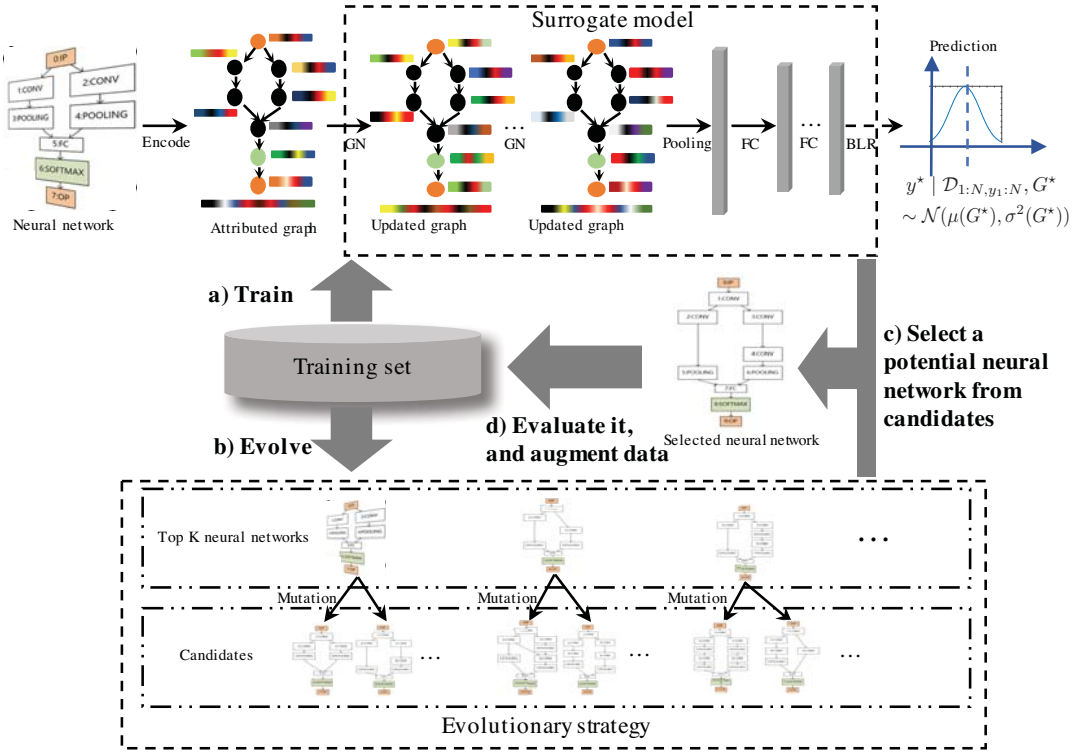

Figure 1: The workflow of the proposed NASGBO method.

actively selects the most potential samples for real-world evaluations based on fitting results, and can quickly get the optimal location of the objective with only a few attempts. BO takes advantage of the information from the previous evaluations to update the quality of the surrogate model, which in turn, will help acquisition function to make better decision about where to evaluate next time. This is the main rationale why BO works more efficient. BO has a large number of applications including hyperparameter tuning for machine learning models [5, 17].

BO can be applied to improve the EA-based neural architecture search methods. By designing a good surrogate function, which can appropriately fit the classification or regression accuracy of a given neural architecture, BO can help EA-based method find an approximately optimal architecture by evaluating only a few candidates, and thus greatly reducing the computation overhead and saving GPU time. The current methods use Gaussian processes (GPs) as surrogate function, define the similarity between two neural architecture according to their structure difference, and then construct a similarity matrix as the kernel matrix of GPs [8, 9]. Because the function mapping the structure of a neural architecture to its predictive performance is black box, it is difficult to know which features are appropriate or even relevant. The features that affect prediction may be structural features, attributes of nodes, global features of networks, or even some implicit features. Therefore, it is difficult for the man-made similarity matrix to reflect the characteristics of neural architecture in an all-round way, which maybe responsible for the failure of finding optimal solution. In addition, these manually customized kernel matrices contain subjective factors, and different definitions often lead to different results. Moreover, GPs is not scalable. Both the training and prediction of GPs involve computing the inverse of kernel matrix, with a time complexity of $O(N^3)$. When the number of observation samples N becomes large, training and prediction become very slow, making it impossible to explore a larger search space.

In view of these problems, we plan to propose a new surrogate that can automatically extract useful features from neural architectures, and these features can be used to fit the mapping from architecture to predictive performance. Specifically, we model neural networks as attributed graphs and model the task of neural architecture search as the task of attributed graph optimization. Instead of GPs, we use Bayesian graph neural network (GNN) as new surrogate. GNN is a deep model of graph representation learning, which supports supervised learning of node and link embeddings from the context of attributed graphs. Its parameter sharing mechanism can greatly drop model complexity, which can not only reduce the time of training and prediction, but also avoid over-fitting. These advantages of GNN are especially suitable for the BO in which less training data are available. After embedding each layer, the representation of overall neural architecture can be obtained by pooling operations.

Based on the new surrogate, we develop a graph Bayesian optimization framework to address the problem of attributed graph optimization. We name it NASGBO, i.e., Neural Architecture Search with Graph Bayesian Optimization. Figure 1 illustrates its workflow. The proposed surrogate is given at the top. It consists of a GNN layer, a pooling layer, a MLP layer and a BLR layer. The input of GNN is the attributed graph encoding an input neural architecture, and its output is the embedding of nodes and links. Pooling layer combines node/link embeddings into the representation of the entire attributed graph and outputs it to MLP layer. MLP layer includes multiple fully connected layers, and predicts the predictive accuracy of the input neural architecture according to its embedding. In order to capture the uncertainty of MLP prediction, we add a BLR (Bayesian linear regression) layer. Note that we only add uncertainty at the last layer of the surrogate model, rather than modeling all model parameters as random variables. This is to balance the need of uncertainty measuring and the cost of computation.

By randomly generating, training and testing some neural architectures, we first prepare an initial training set. The training set is used to train the surrogate model (step a). Based on the current training set, a new population is generated by evolutionary operations (step b). The surrogate is used to predict the performance of each new individual, and one potential individual is selected out of them by maximizing the acquisition function (step c). Then the individual is trained, tested, and added to the current training set (step d). Under some cost constraints, the process is repeated until an approximate optimal solution is returned. Each component of the framework will be elaborated in remaining text.

2 PROBLEM STATEMENT

At present, there are two main ways to describe the architecture of neural networks [3]. One is multi-branch chained architecture [9] and the other is cell-based architecture [13, 15]. In the former representation, each layer can choose different operations such as pooling, convolution, etc. The output of each layer can be used as the input of all subsequent layers, not just as the input of the next layer. Resnets and Densenets belong to this type. This way is more flexible and can generate any kinds of network architectures, so the search space is correspondingly very large. In the latter representation, a neural network is constructed by cells as building blocks, e.g., normal cell and reduction cell. Because the structures of cells are fixed, one only needs to optimize the graphs of cells, so cell-based representation can greatly reduce the search space and allow us to generate deeper network architectures. Limited by space, this work only focus on the multi-branch chained architecture as an example to demonstrate and verify our framework because it has larger search space with more challenges. It is worth noting that our proposed framework can be readily applied to the cell-based architecture by concentrating cells into nodes.

2.1 Objective Formula

Let $f: Q \rightarrow R$ be the performance evaluation function of neural architecture, where Q denotes search space and R denotes predictive accuracy. The goal of neural architecture search is to find the architecture with the highest predictive accuracy, so the objective function can be formalized as:

$$G^{\star} = \arg\max_{q \in Q}[f(q)]$$

Each architecture in search space is modeled as an attributed graph $q = \{V, E, F_V, F_G\}$, where V is a set of nodes denoting the layers of neural architecture, E is a set of edges, F_V is the feature set of nodes and F_G is the global feature set of the graph.

2.2 Design of Search Space

One of the key points is how to construct the neural network architecture search space Q. The search space defines the variables of the optimization problem and is also different corresponding to different search algorithms. In order to apply the GNNs method to Bayesian optimization, we use attributed graph to represent each neural network architecture. We encode the neural network into an attributed graph as input, which can fully exploit various features of the architecture. The attributed graph is mainly composed of network structure, layer attributes, and global attributes of the architecture. The network structure consists of a layer set L and a directed edge E. The directed edge (u, v) indicates whether the output of layer node u is the input of the next layer v. Each node here is treated as a layer.

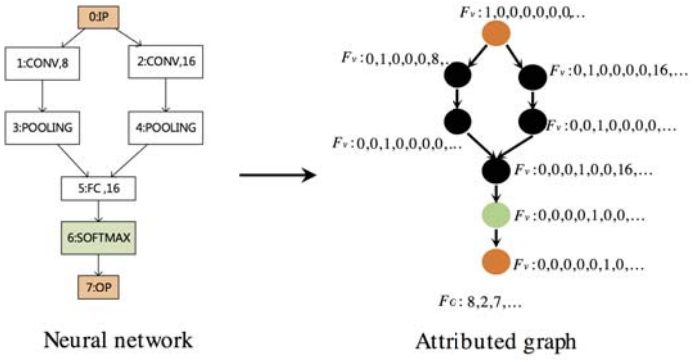

Figure 2: Representing a CNN as an attributed graph. F_v and F_G denote the node and global attributes, respectively.

Taking CNNs as an example, each layer is a node. Node attributes include layer types (softmax, conv3, conv5, conv7, res3, res5, res7, fc, max-pool, avg-pool, ip, op, etc), which each layer type occupys a single dimension in the node features. Other node attributes include the number of convolution units, the number of fully connected units, etc. The global graph feature include the num of the various node types, the num of nodes, the num of edges, etc. The connected edges between nodes represent the output of upper layer as the input of next layer. The index number of the node is obtained by an topological ordering, that is to say, a node with a smaller index number always points to a node with a larger index number.

Figure 2 provides an illustration from a CNN to the corresponding attributed graph. there are 8 nodes in the figure. To simplify the description, only first few dimensions of node attributes have been shown here, i.e. ip, op, cov3, pooling, fc, softmax, op and the

number of units. For example, the attributes of node 2 include cov3 and the number 16 of units.

3 THE COMPONENTS OF NASGBO FRAMEWORK

As illustrated by the Figure 1, our proposed NASGBO framework consists of three main components: deep surrogate model, acquisition function and evolutionary strategy. The workflow of NASGBO is described in Algorithm 1.

Algorithm 1 NASGBO

Input: Neural network architecture initialization sets $G_0 = \{g_1, g_2, ..., g_m\}$, the architecture of GNNs surrogate model SM, hyper-parameter sampling S, iterations Num

Output: Optimal neural network architecture $g_{\max}$ and the performance y of test set

1: After the initialized neural networks G_0 are trained, get the performance y_i of validation set, then integrate into $D = \{(g_1, y_1), (g_2, y_2), ..., (g_m, y_m)\}$, $G = \{g_1, g_2, ..., g_m\}$.
2: **for** $t = 1, 2, ..., Num$ **do**
3: Train SM with training set D
4: Sampling S hyper-parameter samples from their posterior distribution $p(\theta \mid D)$
5: G generates n individuals $G_{next} = \{g_1, g_2, ..., g_n\}$ by evolutionary strategy
6: Select a neural network architecture g_{next} from G_{next} by maximizing EI value
7: After the g_{next} is trained, get performance y_{next} of validation set, then augment data $D = D \cup \{(g_{next}, y_{next})\}$, $G = G \cup \{g_{next}\}$.
8: **end for**
9: Training the neural network architecture $g_{\max}$, where $g_{\max}$ has the best performance, get the performance y of test set
10: **return** $\{g_{\max}, y\}$

3.1 Bayesian GNN surrogate

Due to the issues of the GPs as mentioned in the introduction, we use a deep graph neural network as the surrogate function rather than GPs. In addition, to make our surrogate more scalable and be able to model uncertainty we integrate a layer of BLR (Bayesian linear regressor).

Graph Neural Networks (GNNs) is a deep learning method designed for graph data [1], which can be used to learn the representation of an attributed graph. GNN generalizes various neural network methods for manipulating graphs and defines a class of functions for relational reasoning based on graph structure representations. GN (graph network) module is the main computational unit of the GNN, which is a "graph to graph" module that takes the graph as the input and returns a graph as its output. The nodes and edges have their own attributes, which can be a vector or tensor.

In our framework, a GN block that we adopt contains three "update" functions:

$$e'_k = MLP_e([e_k, v_{r_k}, v_{s_k}, u])$$
$$v'_i = MLP_v([\overline{e'_i}, v_i, u])$$
$$u' = MLP_u([\overline{e'}, \overline{v'}, u])$$

where $\overline{e'_i} = sum(E'_i)$, $\overline{e'} = sum(E')$, $\overline{v'} = sum(\overline{V'})$, MLP_e, MLP_v and MLP_u have five layers, sum is an element-wise sum operation, u represents a vector of the global attributes, V is a set of nodes, each V_i represents the attributes of the node i, E is a set of edges, where each e_k represents the attribute of the edge k, r_k is the index of the receiving node, and s_k is the index of the sending node, $E'_i = \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$, and $V' = \{v'_i\}_{i=1:N^v}$. The edge attribute is updated with the node attribute connected to the edge, the node is updated with the edge attribute connected to the node.

Using the pooling layer, we make full use of the node features, the edge information of the graph and the global features to learn the global representation of the whole graph.In order to capture the uncertainty of the graph when predicting the metric of the graph, we add a Bayesian linear regression (BLR) as the last layer of the surrogate architecture behind multiple fully connected layers. We refer to this model as adaptive basis regression, which are parameterized by the weights and biases of the deep neural network. The form of the BLR is as follows:

$$y_{1:N} = \Phi(.)^T w + b$$

where y is the output of the surrogate function and $b \sim N(0, \sigma^2_{noise} I)$, which is normal distribution, and $\Phi(.)$ is the decision matrix output by previous layers as the input of BLR layer. Given a prior distribution on weights: $w \sim N(0, \sigma^2_w I)$, where σ^2_w denotes the uncertainty of w.

The measure of $G^\star$ can be predicted by:

$$y^\star \mid \mathcal{D}_{1:N}, y_{1:N}, G^\star \sim \mathcal{N}(\mu(G^\star), \sigma^2(G^\star))$$

where $\mathcal{D}_{1:N}$ are observations, $y_1 : N$ are evaluated measures,

$$\mu(G^\star) = \sigma^{-2}_{noise} \Phi(G^\star)^T K^{-1} \Phi(.)_{y_{1:N}},$$
$$\sigma^2(G^\star) = \Phi(G^\star)^T K^{-1} \Phi(G^\star) + \sigma^2_{noise},$$
$$K = \sigma^{-2}_{noise} \Phi(.)\Phi(.)^T + \sigma^{-2}_w I.$$

For Bayesian optimization, if surrogate function is the GPs, Maximizing the acquisition function takes $O(N^3)$ time to calculate the inverse of surrogate matrix $(N \times N)$.For Deep Graph Bayesian Optimization, it takes $O(M^2N)$ and $O(M^2)$ time to maximize the acquisition function. M is the number of units on BLR layer(usually $M \ll N$). Therefore, the Deep Graph Bayesian Optimization takes less time when maximizing the acquisition function.

3.2 Acquisition function

In Bayesian optimization, it can be seen that the probability description of objective function f can be quantified by sampling. Usually the acquisition function mainly uses exploiting and exploring sampling ideas. Exploring new spaces helps to estimate a more accurate objective function f, and sampling near the existing results (usually near the maximum) are expected to find a larger one. The purpose of the acquisition function is to balance the two sampling ideas and quantify the potential of candidate graphs based on previous validations. Given a graph search space $\mathcal{G}$ and a hyper-parameter space Θ, we can define the acquisition function $\mathcal{U} : \mathcal{G} \times \Theta \to \mathbb{R}$. Although any other acquisition functions can be used in the proposed framework, this paper uses expected improvement (EI), a

simple, valid, and common criterion, as the acquisition function. The EI function is the expectation of the improvement function $I(\boldsymbol{x}^*) = max\{0, (\mu(\boldsymbol{x}^*) - y_{max})\}$ at candidate point $\boldsymbol{x}^*$. Specifically, it can be fulfilled by

$$\mathcal{U}(\boldsymbol{x}^*|\mathcal{D}_t, \theta) = (\mu(\boldsymbol{x}^*) - y_{max})\Phi(z(\boldsymbol{x}^*)) + \sigma(\boldsymbol{x}^*)\phi(z(\boldsymbol{x}^*)),$$

where $z(\boldsymbol{x}^*) = \frac{\mu(\boldsymbol{x}^*) - y_{max}}{\sigma(\boldsymbol{x}^*)}$, y_{max} is the maximum value in the current set of observations $\mathcal{D}_t$, and $\Phi(.)$ and $\phi(.)$ denote the cumulative distribution function and probability density function of the standard normal distribution, respectively.

3.3 Evolutionary strategy

Unlike NASBOT and NASNM, We firstly use an evolutionary algorithm (EA) to generate neural networks of k generations rather than optimising the acquisition function . The value of k may be 1, 2, etc. Then we use deep graph bayesian optimization to select the most potential neural network . However , by using EA to optimise the acquisition function , the selected neural network may be very deep and lead to resource crunch after many generations of mutation. This is the main reason why we use EA only for generating. The above is just a simple strategy to balance the breadth and depth of mutation. It can also be tackled in other good strategies.

Each neural network is a feasible solution in the search space, and an approximate evaluation is needed to judge the performance of neural network. We encode each neural network model into a graph.

The mutation operation of a neural network model can be transformed into operation on the graph, such as adding a node, deleting a node, adding an edge, etc. For example, adding a convolution layer is equivalent to adding a node to the graph. The training environment and mutation operations of NASGBO are the same as NASBOT. We only uses mutation operations to generate new candidates, the optimization is guided by BO. Different solutions will be produced with different initializations. To eliminate this effect, all compared methods use the exactly same initial candidate set. Table 1 shows several types of operations.

Mutation Operation	Description
Adding skip	Randomly select two convolution layer ids for skip connection
Incresing units	Increase the number of units by 1/8
Decresing units	Decrease the number of units by 1/8
Adding layer	Randomly select two convolution layer ids for adding a convolution layer
Removing layer	Randomly select a convolution layer id for removing

Table 1: Descriptions of neural architecture mutation operations

4 EXPERIMENTS AND ANALYSIS

To evaluate the performance of the proposed Neural Architecture Search with Graph Bayesian Optimization (NASGBO), we compare it with the following baselines. We mainly verify the performance of NASGBO from two perspectives including accuracy and cost.

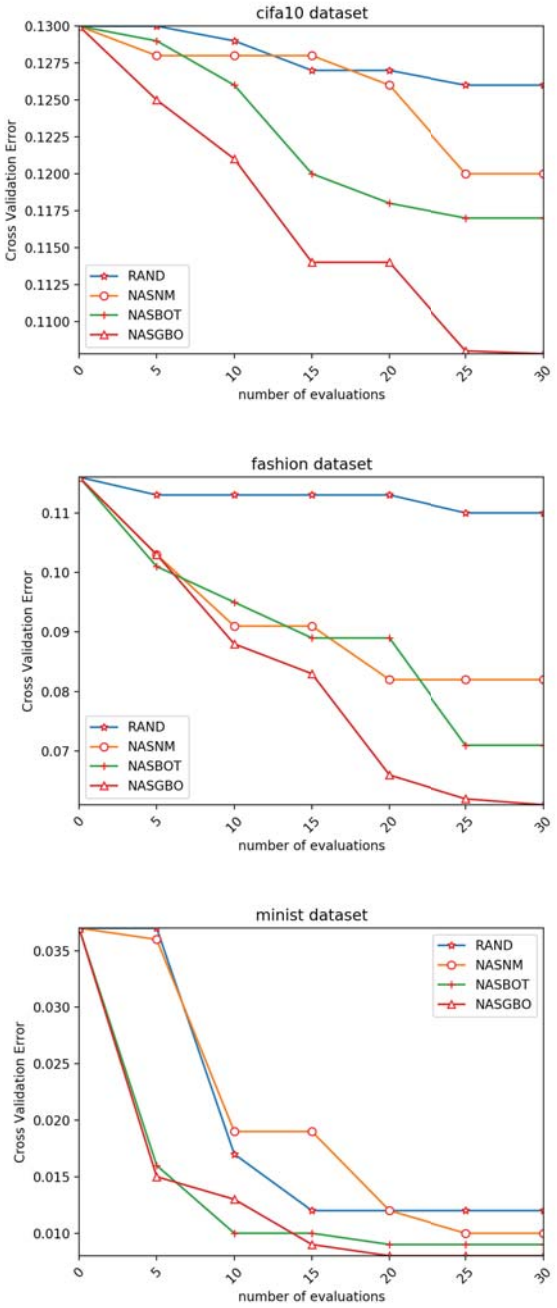

Figure 3: The best validation score for each method against the number of evaluations for CNNs

4.1 Baseline Algorithms

- **RAND**: In the case of the same initial individuals, rand algorithm selects one from initial individuals for mutation and evaluation.
- **TreeBO [7]**: A BO method that only searches over feedforward structures.
- **SEAS [2]**: A method to efficient architecture search for convolutional neural networks based on hill climbing.
- **NASNM [8]**: A Bayesian optimization algorithm in which the network morphism is used to construct kernel function.
- **NASBOT [9]**:A Bayesian optimization algorithm for neural architecture search. Firstly, NASBOT utilizes Optimal Transport Metrics for Architectures of Neural Networks to represent the similarity of the networks, and then searches neural network architectures through the GPs.

Unlike NASBOT and NASNM, we use graph neural network (GNNs) as the surrogate function. It can adequately represent the architectural features of the neural networks.

4.2 Dataset

We use five data sets for the experiments, as follows: Indoor Location [4], Slice Localisation [6], Cifar10 [10], Minist [11], Fanshion Minist [18].

The first two data sets are applied to the regression problem of MLPs. The last three data sets are applied to the classification tasks of CNN images.

For the first two data sets, we split the data sets using a scale of 0.6-0.2-0.2, which used as training data sets, test data sets, and validation data sets respectively, and normalize these data sets to have zero mean and unit variance.For the Cifar10 data sets, there are 60,000 images, of which 50,000 are for training and 10,000 are for testing. We used a 40K-10K-10K ratio to segment the data sets, which used as training dataset, test dataset and validation dataset.

For the last two data sets, there are 70,000 images, of which 60,000 are for training and 10,000 are for testing. We used a 50K-10K-10K ratio to segment the data sets.

4.3 Results

Our method is executed on a single GPU (NVIDIA GeForce GTX 1080 Ti). The regression MSE or classification error (lower is better) on the test set is selected as the evaluation metric.The surrogate will be retrained at each iteration. It only takes 25s to retrain the surrogate and select next potential architecture. Note, the cost is far less than the average cost of evaluating (i.e. training) a neural architecture (e.g., 0.5 hour on Cifar10). For evaluating a network, we provide two different hyperparameters for CNNs and MLPs. For CNNs, hyperparameters are : $weightdecay = 2 * 10^{-4}$, $iterations = 40,000$, $learningrate = 5 * 10^{-3}$(reduced gradually), $batchsize = 32$, and $epsilon = 10^{-5}$. For MLPs, hyperparameters are : $learningrate = 10^{-5}$, $batchsize = 256$, and $iterations = 20,000$. For surrogate model, hyperparameters are :$iterations = 800$, $learningrate = 10^{-4}$, $epochs = 800$, $weightdecay = 10^{-5}$. We adopt the same mutation strategy as NASBOT, and neither method has crossover. Both adopt the same mutation rate (0.5, 0.25, 0.125, 0.075, 0.05), which means to perform one mutation operation with probability 0.5, two operations with probability 0.25, etc.

For Cifar10, Indoor and Slice, all initial networks is consistent with those in the NASBOT paper. Since NASBOT does not provide minist and fashion experiment, we finally choose the three simplest networks as initial networks only have feed forward structure.

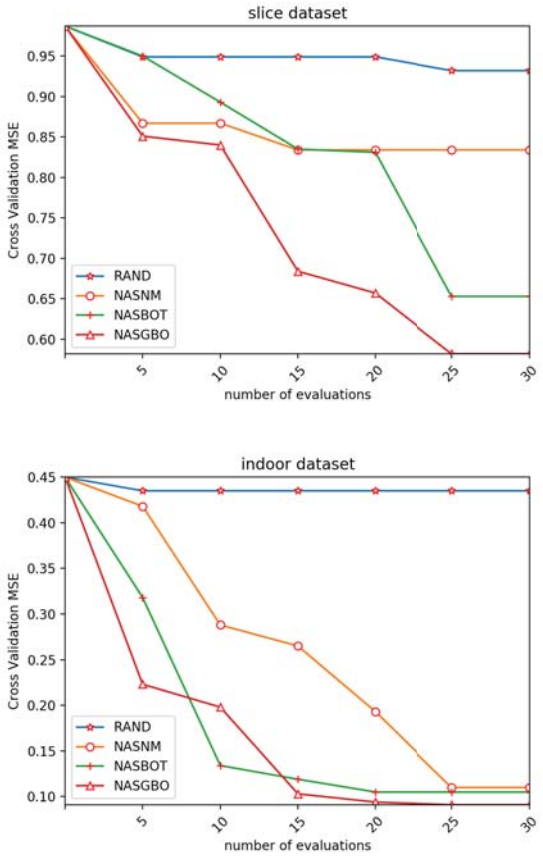

Figure 4: The best validation score for each method against the number of evaluations for MLPs

After running for a period of time (such as 12 hours), the corresponding accuracy of neural network may not increase in the next period of time. Finally, we train the best neural network on the test set to get results of test set.

Table 2 shows that the results on the test set with the best model. As shown by Table 2, NASGBO can get the best result of test set on Cifar10, Fashion, Mnist, and Slice data sets. NASBOT can get the best result on Mnist and Indoor data sets. Relatively speaking, NASGBO performs better than other algorithms in getting the best network architecture.

We use the number of evaluations as cost metric, which does not contain the number of initialized neural networks. The training environment and mutation operations of NASGBO is consistent with NASBOT. We implemented NASBOT conscientiously, and use it as comparative experiment against the number of evaluations on the five data sets .Rand and EA algorithm can be implemented easily

Method	Cifar10	Fashion	Minist	Indoor	Slice
RAND	0.145	0.1100	0.012	0.156	0.932
TreeBO	0.153	–	–	0.168	0.759
SEAS	0.197	0.0800	0.013	–	–
NASNM	0.123	0.0757	0.010	0.112	0.870
NASBOT	0.122	0.0761	0.009	0.114	0.615
NASGBO	**0.120**	**0.0670**	**0.008**	**0.090**	**0.560**

Table 2: The subsequent rows show the regression MSE or classification error (lower is better) on the test set for each method. For TreeBO and SEAS, we choose the best results of test set from NASNM. "–" indicates that the program did not provide an experiment.

compared to our algorithm. We also add them to the comparative experiment.

Figure 3 shows best validation score for each method against number of evaluations for CNNs. As shown by the Figure 3, NASGBO converges faster than NASBOT on Cifar10 and Fashion data sets in terms of the number of evaluations. The performance of two algorithms is almost the same on Minist data sets.

Figure 4 shows best validation score for each method against number of evaluations for MLPs. As shown by the Figure 4, NASGBO converges faster than NASBOT on slice. For the Indoor data set, NASBOT is slightly better than the algorithm we proposed.

For CNNs and MLPs, we analyze the results of optimal architectures we obtained through fashion dataset and slice dataset. Figure 5 shows optimal architectures of NASNM, NASBOT and NASGBO on fashion dataset and Figure 6 shows optimal architectures of NASNM, NASBOT and NASGBO on slice dataset.

As shown by the Figure 5, NASNM uses 5 residual blocks, NASBOT does not use any residual block, and NASGBO uses two residual blocks. From the overall perspective, we get a simpler architecture. According to table 2, we can see that classification error on the test set is lower.

For the slice dataset, the initialized architectures are straight-chain without skip connections. From the results, main reasons for affecting the results of slice dataset are the size of architecture and the skip connection of architecture. As shown by the Figure 6, NASNM can learn about the impact of skip connections on the architecture, and add many skip connections. However, as the size of the architecture grows, the results get worse. NASBOT can also learn a simpler architecture than NASNM, but it is still relatively complicated compared to the architecture we have learned and the value of RMSE is higher than ours. With the use of skip connections, our algorithm makes the architecture as simple as possible. Finally, we can get better result on slice dataset from table 2.

Overall, our method can achieve competitive accuracy with less overhead for on MLPs and CNNs, the two benchmark neural architecture search tasks.

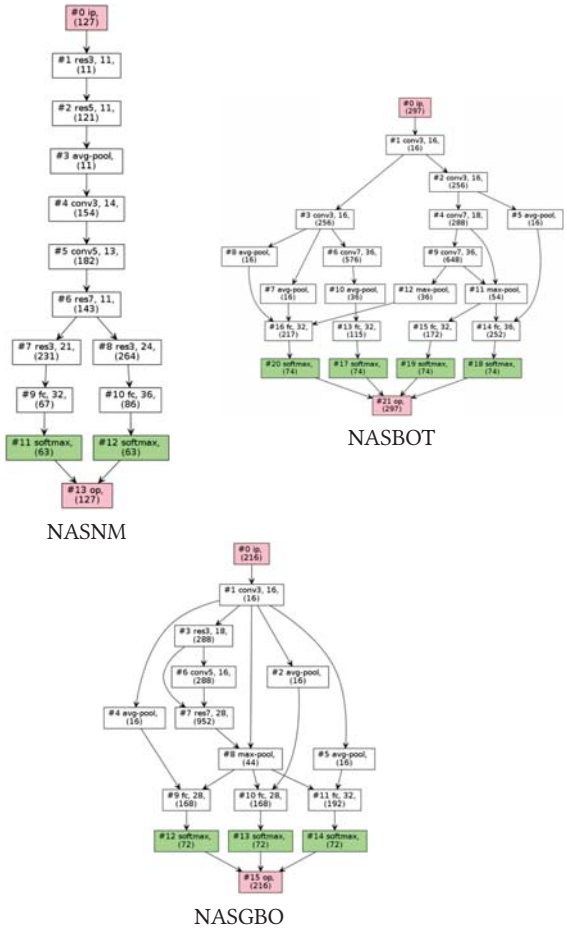

Figure 5: Optimal architectures of NASNM, NASBOT and NASGBO on fashion dataset

5 CONCLUSION

The main contributions of this work are two-fold. (1) We model neural networks as attributed graphs and model the task of neural architecture search as the task of attributed graph optimization. Based on this idea, we develop a graph Bayesian optimization framework to address neural architecture search, which integrating the

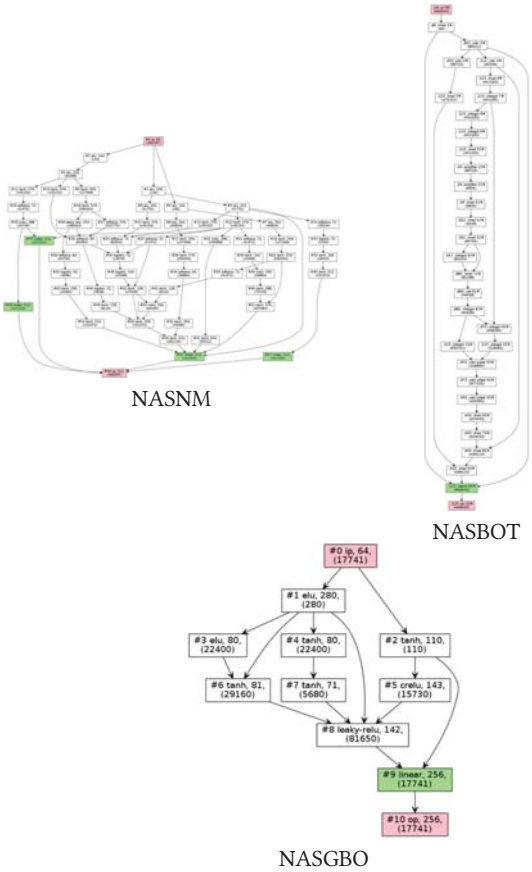

Figure 6: Optimal architectures of NASNM, NASBOT and NASGBO on slice dataset

advantages of EA-based methods and BO-based methods by designing a new surrogate model based on Bayesian GNN. This new surrogate can automatically extract features from deep neural architectures, and can use such features to fit and characterize the black-box mapping from architecture to prediction as well as its uncertainty. Moreover, the training and prediction time of Bayesian optimization can be reduced to linear time by using the Bayesian GNN surrogate, instead of the cubic time of Gauss process, which make our method be able to explore much larger search space and find better solutions. (2) We rigorously show that our method outperforms the state-of-the-art works on benchmark neural architecture search tasks.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China under grants 61572226 and 61876069, and Jilin Province Key Scientific and Technological Research and Development Project under grants 20180201067GX and 20180201044GX.

REFERENCES

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[2] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. 2017. Simple and efficient architecture search for Convolutional Neural Networks. *arXiv preprint arXiv:1711.04528* (2017).

[3] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).

[4] Torres-Sospedra .etc. 2014. UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems. In *Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on*. IEEE, 261–270.

[5] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*. 2962–2970.

[6] Franz Graf, Hans-Peter Kriegel, and Schubert .etc. 2011. 2D image registration in CT images using radial image descriptors. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 607–614.

[7] Rodolphe Jenatton and Archambeau .etc. 2017. Bayesian Optimization with Tree-structured Dependencies. In *International Conference on Machine Learning*. 1655–1664.

[8] Haifeng Jin, Qingquan Song, and Xia Hu. 2018. Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282* (2018).

[9] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. 2018. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*. 2020–2029.

[10] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[11] Yann LeCun and Bottou .etc. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[12] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.

[13] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.

[14] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436* (2017).

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[16] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2018. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548* (2018).

[17] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 847–855.

[18] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[19] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).