# MULTI-OBJECTIVE NEURAL ARCHITECTURE SEARCH VIA PREDICTIVE NETWORK PERFORMANCE OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Neural Architecture Search (NAS) has shown great potentials in finding a better neural network design than human design. Sample-based NAS is the most fundamental method aiming at exploring the search space and evaluating the most promising architecture. However, few works have focused on improving the sampling efficiency for a multi-objective NAS. Inspired by the nature of the graph structure of a neural network, we propose BOGCN-NAS, a NAS algorithm using Bayesian Optimization with Graph Convolutional Network (GCN) predictor. Specifically, we apply GCN as a surrogate model to adaptively discover and incorporate nodes structure to approximate the performance of the architecture. For NAS-oriented tasks, we also design a weighted loss focusing on architectures with high performance. Our method further considers an efficient multi-objective search which can be flexibly injected into any sample-based NAS pipelines to efficiently find the best speed/accuracy trade-off. Extensive experiments are conducted to verify the effectiveness of our method over many competing methods, e.g. $128.4\times$ more efficient than Random Search and $7.8\times$ more efficient than previous SOTA LaNAS for finding the best architecture on the largest NAS dataset NasBench-101.

## 1 INTRODUCTION

Recently Neural Architecture Search (NAS) has aroused a surge of interest by its potentials of freeing the researchers from tedious and time-consuming architecture tuning for each new task and dataset. Specifically, NAS has already shown some competitive results comparing with hand-crafted architectures in computer vision: classification (Real et al., 2019b), detection, segmentation (Ghiasi et al., 2019; Chen et al., 2019; Liu et al., 2019a) and super-resolution (Chu et al., 2019). Meanwhile, NAS has also achieved remarkable results in natural language processing tasks (Luong et al., 2018; So et al., 2019).

A variety of search strategies have been proposed, which may be categorized into two groups: one-shot NAS algorithms (Liu et al., 2019b; Pham et al., 2018; Luo et al., 2018), and sample-based algorithms (Zoph & Le, 2017; Liu et al., 2018a; Real et al., 2019b). One-shot NAS algorithms embed the architecture searching process into the training stage by using weight sharing, continuous relaxation or network morphisms. However, those methods cannot guarantee the optimal performance of the final model due to those approximation tricks and is usually sensitive to the initial seeds (Sciuto et al., 2019). On the other hand, sample-based algorithms are relatively slower but reliable. They explore and exploit the search space using some general search algorithms by providing potential candidates with higher accuracy. However, it requires fully training of huge amounts of candidate models.

Typically, the focus of most existing NAS methods has been on the accuracy of the final searched model alone, ignoring the cost spent in the search phase. Thus, the comparison between existing search algorithms for NAS is very difficult. (Wang et al., 2019b) gives us an example of evaluating the NAS algorithms from this view. They compare the number of training architectures sampled until finding the global optimal architecture with the top accuracy in the NAS datasets. Besides accuracy, in real applications, there are many other objectives we should concern, such as speed/accuracy trade-off. Hence, in this paper, we aim at designing an efficient multi-objective NAS algorithm to

adaptively explore the search space and capture the structural information of architectures related to the performance.

The common issue faced by this problem is that optimizing objective functions is computationally expensive and the search space always contains billions of architectures. To tackle this problem, we present BOGCN-NAS, a NAS algorithm that utilizes Bayesian Optimization (BO) together with Graph Convolutional Network (GCN). BO is an efficient algorithm for finding the global optimum of costly black-box function (Mockus et al., 1978). In our method, we replace the popular Gaussian Processes model with a proposed GCN model as the surrogate function for BO (Jones, 2001). We have found that GCN can generalize fairly well with just a few architecture-accuracy pairs as its training set. As BO balances exploration and exploitation during searching and GCN extracts embeddings that can well represent model architectures, BOGCN-NAS is able to obtain the optimal model architecture with only a few samples from the search space. Thus, our method is more resource-efficient than the previous ones. Graph neural network has been proposed in previous work for predicting the parameters of the architecture using a graph hypernetwork (Zhang et al., 2019). However, it's still a one-shot NAS method and thus cannot ensure the performance of the final found model. In contrast, we use graph embedding to predict the performance directly and can guarantee performance as well.

The proposed BOGCN-NAS outperforms current state-of-the-art searching methods, including Evolution (Real et al., 2019b), MCTS (Wang et al., 2019b), LaNAS (Wang et al., 2019a). We observe consistent gains on multiple search space for CV and NLP tasks, i.e., NasBench-101 (denoted NasBench) (Ying et al., 2019) and LSTM-12K (toy dataset). In particular, our method BOGCN-NAS is $128.4\times$ more efficient than Random Search and $7.8\times$ more efficient than previous SOTA LaNAS on NasBench (Wang et al., 2019a). We apply our method to multi-objective NAS further, considering adding more search objectives including accuracy and number of parameters. Our method can find $42\%$ Pareto optimal architectures in the optimal Pareto front while Random Search finds none after sampling $2000$ architectures on NasBench. We also apply our algorithm on open domain multi-objective search with a ResNet Style search space and the resulting models can dominate famous ResNets (He et al., 2016). The results of experiment demonstrate our proposed algorithm can find a more competitive Pareto front compared with other sample-based methods.

## 2 Related Work

### 2.1 Bayesian Optimization

Bayesian Optimization aims to find the global optimal over a compact subset $\mathcal{X}$ (here we consider maximization problem):

$$x^* = \arg\max_{x \in \mathcal{X}} f(x). \tag{1}$$

Bayesian Optimization considers prior belief about objective function and updates posterior probability with online sampling. Gaussian Processes (GPs) is widely used as a surrogate model to approximate the objective function (Jones, 2001). And *Expected Improvement* acquisition function is often adopted (Mockus et al., 1978). For the hyperparameters of the surrogate model $\Theta$, we define

$$\gamma(x) = \frac{\mu(x; \mathcal{D}, \Theta) - f(x_{best})}{\sigma(x; \mathcal{D}, \Theta)}, \tag{2}$$

where $\mu(x; \mathcal{D}, \Theta)$ is the predictive mean, $\sigma^2(x; \mathcal{D}, \Theta)$ is the predictive variance and $f(x_{best})$ is the maximal value observed. The *Expected Improvement* (EI) criterion is defined as follows.

$$a_{EI}(x; \mathcal{D}, \Theta) = \sigma(x; \mathcal{D}, \Theta)[\gamma(x)\Phi(\gamma(x); 0, 1) + \mathcal{N}(\gamma(x); 0, 1)], \tag{3}$$

where $\mathcal{N}(\cdot; 0, 1)$ is the probability density function of a standard normal and $\Phi(\cdot; 0, 1)$ is its cumulative distribution.

### 2.2 Multi-objective Optimization

Without loss of generality about $max$ or $min$, given a search space $\mathcal{X}$ and $m \geq 1$ objectives $f_1 : \mathcal{X} \to \mathbb{R}, \ldots, f_m : \mathcal{X} \to \mathbb{R}$, variable $X_1 \in \mathcal{X}$ dominates variable $X_2 \in \mathcal{X}$ (denoted $X_1 \succ X_2$) if (i) $f_i(X_1) \geq f_i(X_2), \forall i \in \{1, \ldots, m\}$; and (ii) $f_j(X_1) > f_j(X_2)$ for at least one $j \in \{1, \ldots, m\}$. $X^*$ is *Pareto optimal* if there is no $X \in \mathcal{X}$ that domaines $X^*$. The set of all Pareto optimal architectures consitutes the *Pareto front* $\mathcal{P}_f$. A multi-objective optimization problem (MOP) aims at finding such

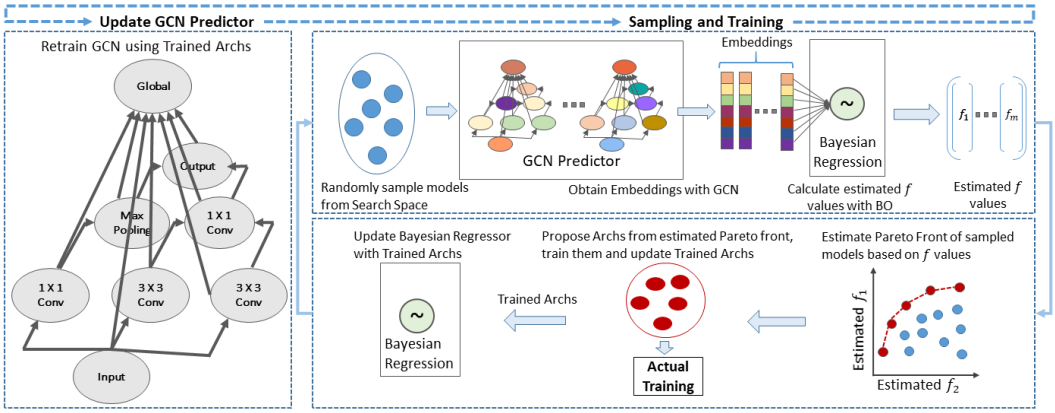



Figure 1: The overview of our proposed algorithm. BOGCN-NAS is integrated by GCN and Bayesian Linear Regression, including two iterative phases: 1) Sampling and Training; 2) Update GCN predictor. During phase-one, we randomly sample a group of architectures from the search space and calculate a predicted objective $f$. Then an estimated Pareto front is constructed by $f$ and Pareto optimal points are selected as proposed architectures. Then we fully train them and add into trained architecture set ($U$). The Bayesian regression is updated with $U$. During phase-two, we further retrain the GCN predictor with $U$.

input $X \in \mathcal{X}$ that $X$ cannot be dominated by any variable in $\mathcal{X}$ (Marler & Arora, 2004). With the found Pareto front $\widetilde{\mathcal{P}_f}$, the overlap $|\widetilde{\mathcal{P}_f} \cap \mathcal{P}_f|/|\mathcal{P}_f|$ can be used to measure the performance of the search method (Okabe et al., 2003).

### 2.3 Graph Convolutional Network

Let the graph be $G = (V, E)$, where $V$ is a set of $N$ nodes, and $E$ is the set of edges. Let its adjacency matrix be $A$ and feature matrix be $X$. The graph convolutional network (GCN) is a learning model for graph-structure data (Kipf & Welling, 2016). For a $L$-layer GCN, the layer-wise propagation rule is given by:

$$H^{(l+1)} = f(H^{(l)}, A) = \mathrm{ReLU}(\widetilde{D}^{\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}), \tag{4}$$

where $\widetilde{A} = A + I$, $I$ is the identity matrix, $\widetilde{D}$ is a diagonal matrix with $\widetilde{D}_{ii} = \sum_{j=1}^{N} A_{ij}$, $H^{(l)}$ and $W^{(l)}$ are the feature map and weight at the $l$-th layer respectively, and $\mathrm{ReLU}(\cdot)$ is the ReLU activation function. $H^{(0)}$ is the original feature matrix $X$, and $H^{(L)}$ is the graph embedding matrix.

## 3 BOGCN-NAS

To search for the optimal architecture more efficiently, we proposed BOGCN-NAS by using predictive network performance Optimization with the GCN (Section 3.2) while utilizing the Bayesian Optimization. Figure 1 shows the overview of the proposed algorithm.

### 3.1 Multi-objective NAS

We formulate NAS problem as a multi-objective optimization problem over the architecture search space $\mathcal{A}$ where objective functions can be accuracy, latency, number of parameters, etc. We aim to find architectures on the Pareto front of $\mathcal{A}$. Specifically, when $m = 1$, it reduces to single-objective (usually accuracy) NAS.

### 3.2 GCN Predictor

GCN predictor predicts the performance (like accuracy) of an architecture. Compared with MLP and LSTM predictors proposed before (Wang et al., 2019b), GCN can preserve the context of graph data better. Another important characteristic of GCN is its ability to handle variable number of nodes, while an MLP cannot take a larger architecture as the input. Even though the LSTM can handle variable-length sequences, its performance is not competitive compared with other two models.

A neural network can be viewed as a directed attributed graph, in which each node represents an operation (such as convolution operation) and each edge represents a data flow. As a concrete illustration, we use the architectures in the NasBench dataset (Ying et al., 2019) as an example. The idea can be easily extended to other architectures. In NasBench, each architecture is constituted by

stacking multiple repeated cells. Thus, we will focus on searching the cell architecture. An example cell in NasBench is illustrated on the left side of Figure 1, where "input" represents the input of the cell, "output" represents the output of the cell, "$1 \times 1$ Conv, $3 \times 3$ Conv, Max Pooling" are three different operations (5 operations totally).

We propose to encode the cell into an adjacency matrix $A$ (asymmetric) and a feature matrix $X$, as the input of our GCN predictor. Note that the vanilla GCN only extracts the node embeddings, while we want to obtain graph embedding. Following (Scarselli et al., 2008), we add a global node to the original graph of the cell and let every node point at the global node. The adjacency matrix can be obtained directly from the graph structure. For the feature matrix, we use the one-hot coding scheme for each operation. Besides the original 5 different operations defined in NasBench, we add another operation (global node) into coding scheme.

We feed $A$ and $X$ to a multi-layer GCN model to obtain the embedding of every node $H^{(L)}$ by (Eq. 4). For high-level prediction, we leave original nodes out and take the embedding of global node solely because it already has the overall context of the architecture. Followed by one fully-connected layer with sigmoid activation function, we can get the predicted accuracy. In training phase, we use MSE loss for regression optimization.

### 3.3 Incorporating BO into GCN

Bayesian Optimization is an efficient model for search and optimization problems, which considers balancing both exploitation and exploration balanced. It depends on updating the posterior distribution with the samples drawn from the search space based on one cheap surrogate model.

GPs is one of the most popular choices because Gaussian distribution is self-conjugate such that a posterior distribution is also the same form as the prior. However, since the computation increases cubically with the number of samples (Snoek et al., 2015), GPs is not suitable for NAS problem, whose search space is always huge. Another drawback for GPs is that it cannot handle graph-data directly without a special encoding scheme. In this work, we replace the popular surrogate model with our GCN predictor and take the uncertainty of all architectures into consideration.

Inspired by previous work (Snoek et al., 2015), we train the GCN predictor first with the trainied architecture set $\mathcal{D}$ containing architectures $\{(A_i, X_i)\}_{i=1}^n$ with their actual performances $\{t_i\}_{i=1,}^n$, then during searching, we replace the last fully connected layer with Bayesian Linear Regressor (BLR) for Bayesian estimation and retain GCN related layers for point estimation. We only consider the uncertainty of the weights on the last fully-connected layer. We denote the embedding function of global node by $\phi(\cdot,\cdot) = [\phi_1(\cdot,\cdot), \phi_2(\cdot,\cdot), ..., \phi_d(\cdot,\cdot)]^T$. We can get the embedding of every architecture $\phi(A_i, X_i)$ from the trainied architecture set $\mathcal{D}$ and treat them as the basis functions for BO. For clarifying, we define $\Phi$ to be the design matrix where $\Phi_{ij} = \phi_j(A_i, X_i)$.

Different from typical Bayesian Linear Regression (BLR) (Bishop, 2006), the final layer of our GCN model contains non-linear activation function. Here we use the inverse function trick to avoid the non-trivial variant. Without fitting the true accuracy $t$, we prefer to estimate the value before the activation function $logit(t)$ such that we can convert non-linear regression to linear regression problem. The key of BO is the order of the acquisition function over all sampled architectures rather than true values. Due to the monotonicity of sigmoid function, the order property still holds. The predicted mean and variance given by our model without the last activation function are shown below.

$$\mu(A, X; \mathcal{D}, \alpha, \beta) = m_N^T \phi(A, X), \tag{5}$$

$$\sigma^2(A, X; \mathcal{D}, \alpha, \beta) = \frac{1}{\beta} + \phi(A, X)^T S_N \phi(A, X), \tag{6}$$

where

$$m_N = \beta S_N \Phi^T logit(t), S_N = \alpha I + \beta \Phi^T \Phi. \tag{7}$$

Here, $\alpha, \beta$ are precisions of the prior, which are hyperparameters of the Bayesian Optimization model. We can estimate them by maximizing the log marginal likelihood as following Snoek et al. (2012).

$$\begin{aligned} \log p(logit(t)|\alpha, \beta) = \frac{M}{2}\log\alpha + \frac{N}{2}\log\beta - \frac{\beta}{2}||logit(t) - \Phi m_N||^2 \\ - \frac{\alpha}{2} m_N^T m_N - \frac{1}{2}\log|S_N^{-1}| - \frac{N}{2}\log 2\pi. \end{aligned} \tag{8}$$

**Algorithm 1** BOGCN-NAS Search Procedure. $\mathcal{A}$ is the given search space, $l$ is the number of samples every iteration, $k$ is the ratio of $GCN/BLR$ update times, $\mathcal{P}_f$ is the optimal Pareto front and $threshold$ is the criteria for search stopping.

Initialize trained architecture sets $U = \{A_i, X_i, t_i\}_{i=1}^{n}$ from $\mathcal{A}$ and current Pareto front $\widetilde{\mathcal{P}_f}$;
Train $GCN$ and $BLR$ initially;
**while** $|\widetilde{\mathcal{P}_f} \cap \mathcal{P}_f|/|\mathcal{P}_f| < threshold$ **do**
  **for** iteration$= 1, 2, ..., k$ **do**
    $\mathcal{R} = random_sample(\mathcal{A})$;
    $S = sample(GCN, BLR, l, \mathcal{R})$;
    Fully train sampled models $S$;
    Update $U = U \cup S$;
    Update $\widetilde{\mathcal{P}_f}$ in set $U$;
    $BLR.update(GCN, U)$;
  **end for**
  $GCN.retrain(U)$;
**end while**
**return** $\widetilde{\mathcal{P}_f}$

**function** $sample(GCN, BLR, l, \mathcal{R})$
  $embeddings = GCN(\mathcal{A})$;
  Predict mean and variance of models in $\mathcal{R}$ with $BLR$ and $embeddings$ by (Eq. 5) and (Eq. 6);
  Calculate corresponding Expected Improvement by (Eq. 3);
  Sample top $l$ Pareto optimal models sorted by Expected Improvement into set $S$;
  **return** $S$.
**end function**

### 3.4 Search with Alternate Learning

NAS is a process of online learning during which we can utilize new fully-trained architectures sampled from the search space. Therefore, the GCN model and BLR in BOGCN-NAS should be updated with the increasing number of samples for better generalization. In this work, for the reason that GCN retraining is more expensive than BLR updating, we update BLR more frequently than GCN predictor.

The algorithm of our proposed BOGCN-NAS is illustrated in Algorithm 1. Given the search space $\mathcal{A}$, we initialize trained architecture sets $U$ containing architectures $(A_i, X_i)$ with their performance $t_i = \{f_{1i}, \ldots, f_{mi}\}$. We train GCN predictor firstly with U and replace the last fully-connected layer with BLR described in Section 3.3. Then we can calculate every candidate model's *Expected Improvement* as their estimated objective values $\hat{t_j} = \{\hat{f_{1j}}, \ldots, \hat{f_{mj}}\}$. Based on $\hat{t_j}$ and multi-objective formulation (Section 3.1), we can generate a estimated Pareto Front and sample estimated Pareto optimal models as set $\mathcal{S}$ and fully-train them to obtain the true objective values $t_j$. The trained architecture sets is then updated by $U = U \cup S$. Accumulating a certain amount of new data, we update GCN and BLR alternately for next periodic sampling until satisfying the stop criteria.

### 3.5 Exponential Weighted Loss

For GCN predictor alone, MSE loss can achieve competitive performance for fitting all data. However, when it comes to the surrogate function for finding the top-performance model, we should pay more attention to architectures with high performance compared to others.

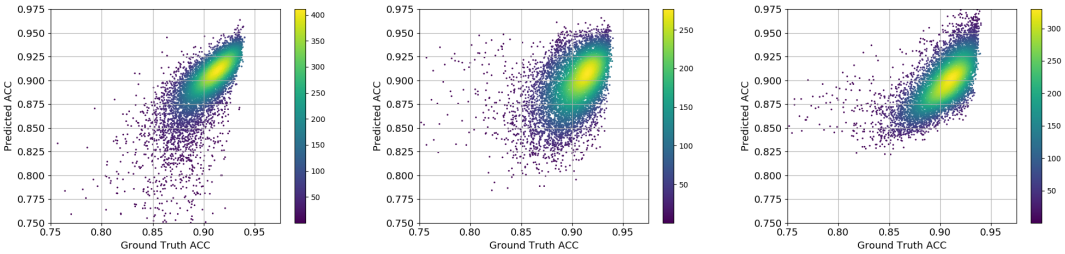
In the search phase of BO, we select samples with top values of acquisition function. Specifically, we expect to predict architectures with high performance as accurately as possible, while distinguishing low-performance architecture is sufficient for our purpose. For adapting NAS task, we propose Exponential Weighted Loss for the surrogate GCN model.

$$L_{exp} = \frac{1}{N(e-1)} \sum_{i=1}^{N} (\exp(\widetilde{y_i}) - 1)||y_i - \widetilde{y_i}||^2, \tag{9}$$

here $y_i$ is the predicted accuracy, $\widetilde{y_i}$ is the ground truth and $e - 1$ is normalization factor ($e$ is the base of the natural logarithm). Thus, our predictor will focus on predicting for those models with higher accuracy.

## 4 Experiment

**Dataset and Search Space** In this Section, we validate the performance of the proposed BOGCN-NAS on NasBench (Ying et al., 2019). This is the largest benchmark NAS dataset focusing on




(a) GCN. (b) MLP. (c) LSTM.

Figure 2: Correlation of predicted accuracy versus ground truth for various predictors on NasBench. It can be found that our GCN predictor has better correlation performance than the competing methods.

computer vision tasks currently, with $420K$ architectures. To show the generalization of our method, we also collect $12K$ LSTM models trained on PTB dataset (MARCUS et al., 1993) aiming at NLP task. We also compare our method in an open domain with ResNet style search space following Li et al. (2019). We will provide the detail of the search space and collection method in Appendices Section.

Specifically, we prove the performance of single proposed GCN Predictor firstly and then validate the whole framework on single-objective/multi-objective search problems compared with other baselines. The results of the experiment illustrate the efficiency of BOGCN-NAS.

### 4.1 Comparison of individual predictors

We use a four-layer GCN, with $64$ hidden units in each layer, as the surrogate model for BO. During training, we use the Adam optimizer (Kingma & Ba, 2014), with a learning rate of 0.001 and a minibatch size of $128$. The proposed GCN predictor is compared with an MLP predictor and LSTM predictor (Liu et al., 2018a; Wang et al., 2019b). For the evaluation metric, we follow (Wang et al., 2019b) and use the correlation between the predicted accuracy and true accuracy. One difference between our evaluation method and (Wang et al., 2019b) is that we compare predictor's performance training on fewer architectures rather than whole search space, because NAS start with little training data and we cannot train such a large number of architectures in reality. We used 1000 architectures in NASBench for training, 100 architectures for validation, and 10000 architectures for testing.

Table 1 shows the correlation result of various predictors and the number of predictors' parameters. The prediction result of three predictors is also illustrated in Figure 2 and the value means the density of architecture around. As can be seen, the GCN predictor can predict the performance of architectures more accurately than the other two predictors. In addition, the GCN predictor has fewer parameters compared with them.

Table 1: Performance of predictors training on small dataset. Our method has much better correlation with less predictor parameters.

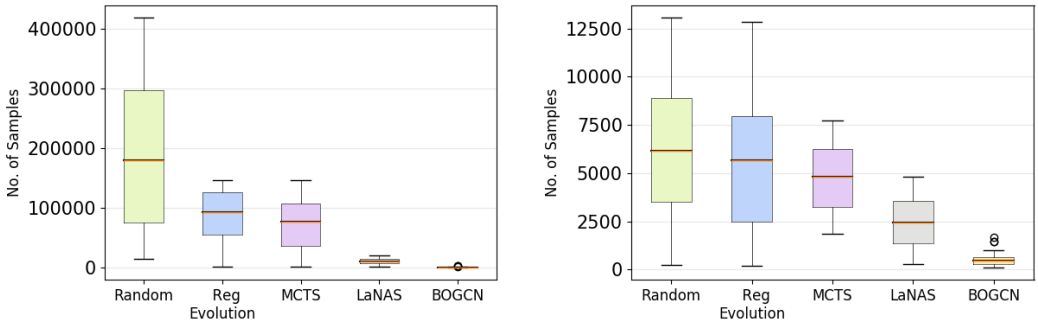
| | Correlation | No. of parameters |
|---|---|---|
| MLP | 0.522 | 6326K |
| LSTM | 0.400 | 92K |
| GCN | **0.819** | **14K** |

Table 2: Comparison of samples required to find the global optimal over 50 rounds.

| | NasBench | LSTM-12K |
|---|---|---|
| Random | 188139.8 | 6182.6 |
| Reg Evolution | 87402.7 | 5670.9 |
| MCTS | 73977.2 | 4687.4 |
| LaNAS | 11390.7 | 2464.4 |
| BOGCN-NAS | **1465.4** | **558.8** |

### 4.2 Single-objective Search

For single-objective (accuracy) search, a special case of multi-objective search. For the proposed BOGCN-NAS, we randomly sample $50$ architectures to fully train and use them as the initial trained architecture sets. During the search phase, we use GCN to obtain embeddings and use the Bayesian regressor to compute EI scores for all architectures in the search domain, rank them based on the score and select the top ten models to fully train, obtain the accuracies and add them to trained architecture sets, update the best accuracy observed. The process is repeated for $k = 10$ times. The GCN predictor is then retrained with our updated trained architecture sets. This is repeated until the target model is found over $50$ rounds.



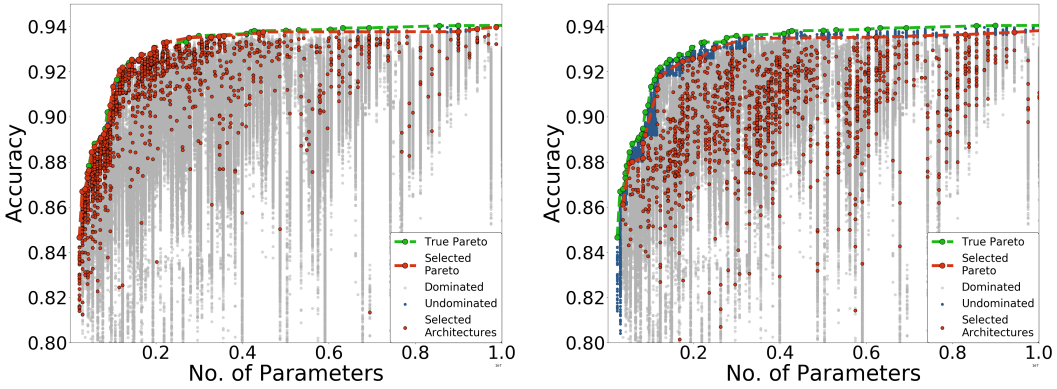

(a) NasBench Dataset (b) LSTM dataset

Figure 3: Comparison of number of architecture samples truly evaluated to find the globally optimal architecture over 50 rounds. On NasBench, BOGCN-NAS is $128.4\times$, $59.6\times$, $50.5\times$, $7.8\times$ more sample-efficient than Random Search, Regularized Evolution, MCTS and LaNAS.

BOGCN-NAS is compared with the following state-of-the-art sample-based NAS baselines: (i) Random Search, which explores the search space without performing exploitation; (ii) Regularized Evolution (Real et al., 2019b), which uses heuristic evolution process for exploitation but is still constrained by the available human prior knowledge; (iii) Monte Carlo tree search (MCTS) (Wang et al., 2019b); and (iv) LaNAS (Wang et al., 2019a). Both MCTS and LaNAS only estimate the performance of models in a coarse subspace, and then select models randomly in that subspace. In contrast, Bayesian optimization conducts a more fine-grained search and predicts the expected improvement of every sample.

Figure 3 shows the number of samples until finding the global optimal architecture using different methods. The proposed algorithm outperforms the other baselines consistently on the two different datasets. On NasBench, BOGCN-NAS is $128.4\times$, $59.6\times$, $50.5\times$, $7.8\times$ more sample-efficient than Random Search, Regularized Evolution, MCTS and LaNAS respectively. On the smaller NLP dataset, BOGCN-NAS can still search and find the optimal architecture with fewer samples.

### 4.3 Multi-objective Search



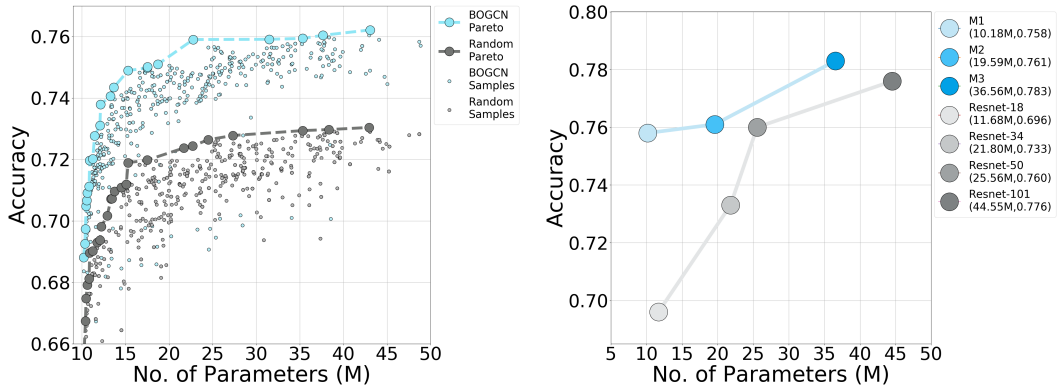

(a) BOGAN-NAS (b) Random Search

Figure 4: The result of multi-objective search with 2000 architecture samples. BOGAN-NAS can find $42\%$ Pareto optimal architectures while Random Search cannot find any.

In this section, we focus on multi-objective (accuracy and number of parameters) search. As the same settings of BOGCN-NAS with Section 4.2, the only difference is the criteria for updating the best architecture observed.

Because few methods consider multi-objective search, here we only compare with Random Search. In detail, we sample $2,000$ architectures in total and compare the found Pareto front $\widetilde{\mathcal{P}_f}$ with the optimal Pareto front $\mathcal{P}_f$on the NasBench dataset.

As shown in Figure 4, the grey dots are the overall architectures of NasBench, the red dots are samples selected by corresponding algorithms and the blue dots are architectures undominated by our selected samples. Based on them, we can make sure the respective Pareto fronts - the green




(a) Comparison of BOGCN-NAS and Random Search. (b) Comparison of the found models and ResNets.

Figure 5: Open domain multi-objective search on ResNet style search space with 500 samples. Compared with Random Sampling, BOGCN-NAS achieves a more competitive Pareto front.

dashed line is the optimal Pareto front and the red dashed line is the estimated Pareto front. The overlap of BOGAN-NAS is $42\%$ while the overlap of Random Search is still $0$ at $2000$ samples.

Compared with Random Search, the models sampled by our method are gathered near the optimal Pareto front, and the found Pareto front is also closer to the optimal Pareto front. From the measure of overlap, our method can find plenty of optimal architectures (covered in optimal Pareto front) with less sampling. This validates the efficiency of our algorithm on this multi-objective search task.

### 4.4 Open domain Multi-objective Search

In this section, we validate the proposed method on an open ResNet style search space (described in Appendices Section) for multi-objective search on ImageNet (Deng et al., 2009). Here we consider classification accuracy and the number of parameters of the model at the same time. With other settings remaining, we train the sampled architectures by early stopping at $40$ epochs rather than fully training.

The accuracy and number of parameters of the sampled model are illustrated in Figure 5. Compared to random sampling, BOGCN-NAS achieves a more competitive Pareto front. We also select three models (M1, M2, M3) on the Pareto front and fully train them. The comparison of our found models and famous ResNet family models are shown in Figure 5b. It shows that ResNets can be dominated by our found models seriously. The final selected architectures can be found in Appendices Section.

### 4.5 Transfer Learning

In this Section, we discover the ability of our proposed algorithm on transfer learning tasks. For validation, we split NasBench into two sub-datasets: architectures with $6$ nodes and architectures with $7$ nodes.

Using the same settings with Section 4.2, we pretrained our GCN model on architectures with $6$ nodes and then transfer the GCN predictor to searching on the architecture domain with $7$ nodes. For comparison, we run the same algorithm without pretraining. The search method with pretrained GCN predictor finds optimal model after $511.9$ samples while the method without pretraining needs $1386.4$ samples. As can be seen, pretraining can reach the optimal model $2.7\times$ more efficiently than unpretrained, validating the transfer ability of GCN predictor. Thus, BOGCN-NAS can handle different-scale architectures as long as their operation choices are the same.

## 5 Conclusion

In this work, we propose BOGCN-NAS, a multi-objective NAS method using Bayesian Optimization with Graph Convolutional Network predictor. We formulate the problem as a multi-objective optimization problem and utilize the efficiency of BO to search top-performance architectures. Instead of using popular Gaussian Processes surrogate model, we replace it with proposed GCN predictor such that graph-based structure can be preserved better. For NAS-specific tasks, we also propose weighted loss focusing on top-performance models. Experimental results show that the proposed algorithm outperforms the SOTA LaNAS on single-objective NAS and validate its efficiency on multi-objective NAS as well.

## REFERENCES


B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2016. C

C. Bishop. *Pattern recognition and machine learning*. 2006. 3.3

H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. C

H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. C

L. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, pp. 8699–8710, 2018. C

Y. Chen, T. Yang, X. Zhang, G. Meng, C. Pan, and J. Sun. Detnas: Neural architecture search on object detection. In *Advances in neural information processing systems*, 2019. 1, C

X. Chu, B. Zhang, H. Ma, R. Xu, J. Li, and Q. Li. Fast, accurate and lightweight super-resolution with neural architecture search. *arXiv preprint arXiv:1901.07261*, 2019. 1

J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, 2009. 4.4

G. Ghiasi, T. Lin, and Q. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7036–7045, 2019. 1

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 1

D. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, pp. 345–383, 2001. 1, 2.1

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014. 4.1

T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016. 2.3

X. Li, Y. Zhou, Z. Pan, and J. Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9145–9153, 2019. 4, D.3

C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018a. 1, 4.1

C. Liu, L. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 82–92, 2019a. 1, C

H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018b. C

H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019b. 1, C

R. Luo, F. Tian, T. Qin, E. Chen, and T. Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pp. 7816–7827, 2018. 1

M. Luong, D. Dohan, A. Yu, Q. Le, B. Zoph, and V. Vasudevan. Exploring neural architecture search for language tasks. In *International Conference on Learning Representations*, 2018. 1

M. MARCUS, B. SANTORINI, and M. MARCINKIEWICZ. Building a large annotated corpus of english: the penn treebank. *Computational linguistics-Association for Computational Linguistics*, pp. 313–330, 1993. 4, D.2

R. Marler and J. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, pp. 369–395, 2004. 2.2

J. Mockus, V. Tiesis, and A. Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, (117-129), 1978. 1, 2.1

T. Okabe, Y. Jin, and B. Sendhoff. A critical survey of performance indices for multi-objective optimisation. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, pp. 878–885, 2003. 2.2

H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pp. 4092–4101, 2018. 1, D.2

E. Real, S. Moore, A. Selle, S. Saxena, Y. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2902–2911, 2017. C

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

E. Real, A. Aggarwal, Y. Huang, and Q. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4780–4789, 2019b. 1, 4.2

F. Scarselli, M. Gori, A. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, pp. 61–80, 2008. 3.2

C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019. 1

J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012. 3.3

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pp. 2171–2180, 2015. 3.3

D. So, Q. Le, and C. Liang. The evolved transformer. In *International Conference on Machine Learning*, pp. 5877–5886, 2019. 1

L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian. Sample-efficient neural architecture search by learning action space. *arXiv preprint arXiv:1906.06832*, 2019a. 1, 4.2, B.1

L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019b. 1, 3.2, 4.1, 4.2, B.1, B.2

S. Xie, H. Zheng, C. Liu, and L. Lin. Snas: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. C

C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114, 2019. 1, 3.2, 4, B.2

C. Zhang, M. Ren, and R. Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2019. 1

B. Zoph and Q. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 1, C

# Appendices

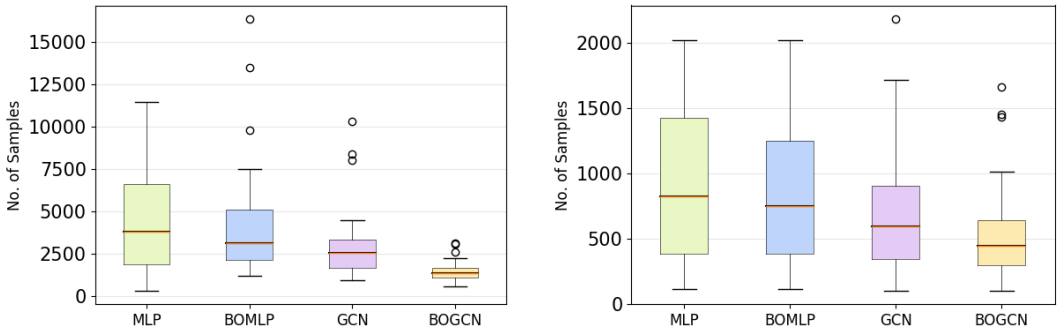
## A Ablation Study

### A.1 The advantage of GCN Predictor

To verify GCN is a superior choice of predictor against others (e.g. MLP, RNN), we replaced GCN in the search algorithm with MLP predictor. All other steps remain the same. The result is shown in Figure 6, GCN is able to discover the optimal architecture with fewer samples than MLP, which proves the superiority of GCN predictor.

### A.2 Effectiveness of Bayesian Optimization

To show that Bayesian Optimization indeed improves the performance of our search algorithm, we remove BO and use point estimation only. In other words, we select candidate models based only on GCN's predictive accuracies of architectures. In this way, we remove the uncertainty introduced by BO and only exploit the current GCN network's predictive accuracy. Figure 6 shows that our algorithm is more efficient with BO.




(a) NasBench (b) LSTM-12K

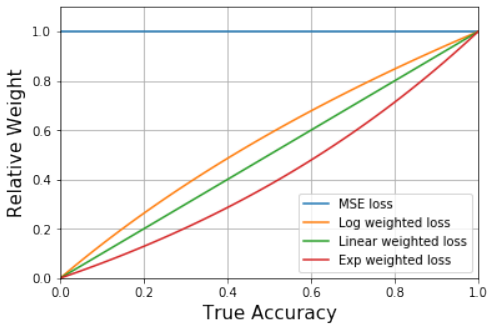
Figure 6: Ablation study of Bayesian Optimization and Graph Convolutional Network Predictor. The comparison shows that the efficiency of the search can be boosted by adding Bayesian optimization and Graph Convolutional Network modules respectively.

### A.3 The improvement of Exponential Weighted Loss

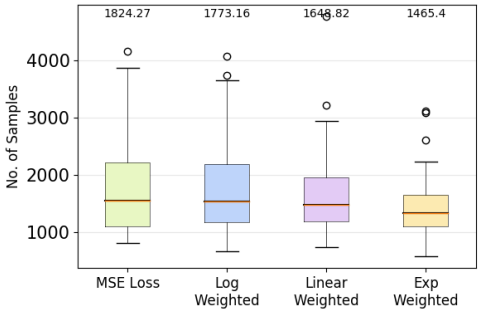
To prove the improvement of our proposed weighted loss empirically, we compare it with MSE loss. We also design other two weighted losses as following to show the influence of the hessian matrix of added weight. The experiment result of single-objective (accuracy) on NasBench is shown in Figure 7. As can be seen, exponential weighted loss outperforms other three losses, which is consistent with our intuition.

$$L_{log} = \frac{1}{N \log 2} \sum_{i=1}^{N} \log(\widetilde{y}_i + 1) ||y_i - \widetilde{y}_i||^2, \tag{10}$$

$$L_{linear} = \frac{1}{N} \sum_{i=1}^{N} \widetilde{y}_i ||y_i - \widetilde{y}_i||^2. \tag{11}$$




(a) The illustration of four losses considered in this paper.



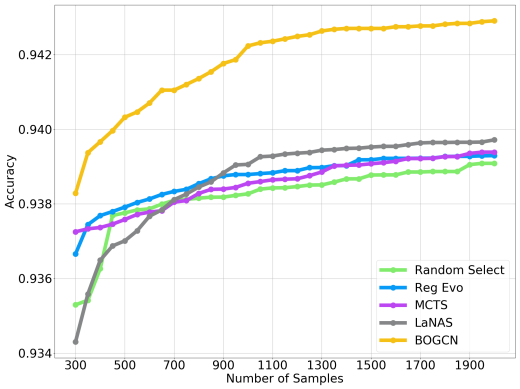

(b) The performance of four losses over 50 rounds. It can be found that our method can reach the highest efficiency with exponential weighted loss.

Figure 7: Ablation study of Weighted Loss.

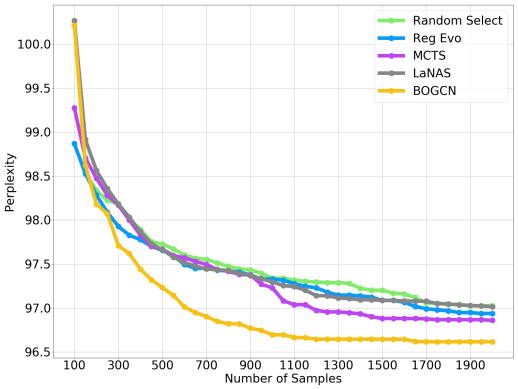
# B SUPPLEMENTAL EXPERIMENTS

## B.1 TIME COURSE PERFORMANCE COMPARISON

Following (Wang et al., 2019b;a), besides comparing the number of training architectures sampled until finding the global optimal architecture, we also evaluate the current best models during the searching process. As shown in Figure 8, our proposed BOGCN-NAS outperform other search algorithms except for the very beginning on LSTM model searching.




(a) NasBench

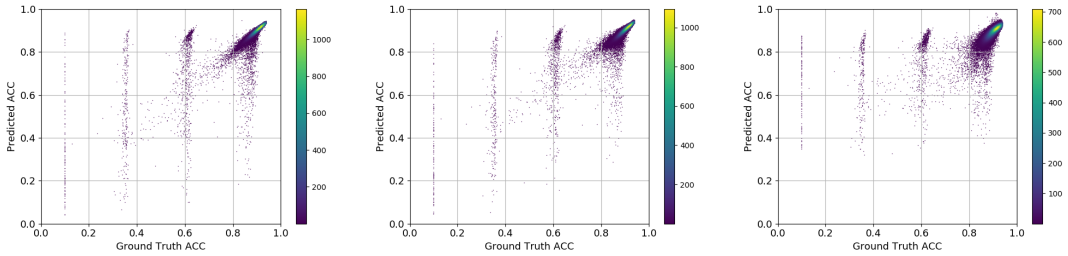



(b) LSTM-12K

Figure 8: Current best accuracy during searching. The result demonstrates that our proposed BOGCN-NAS outperforms other search algorithms significantly.

## B.2 PREDICTOR TRAINING ON WHOLE SEARCH SPACE

For comparison with previous work (Wang et al., 2019b), we also train our GCN predictor on whole NasBench dataset ($420K$ models) (Ying et al., 2019). We used $85\%$ NASBench for training, $10\%$ for validation and remaining $5\%$ for testing. As shown in Figure 9 (the value also means the density of architectures around), GCN outperforms others consistently with experiment training on fewer data (Section 4.1). Even though the correlations of the GCN and MLP are comparable here, the performance is less important than cases training on fewer data.




(a) GCN (corr=0.841). (b) MLP (corr=0.830). (c) LSTM (corr=0.743).

Figure 9: Correlation of predicted accuaracy versus ground truth for various predictors on whole dataset. It shows that GCN predictor can predict more accurately than other two predictors on whole NasBench.

## C NAS Related Work

NAS aims at automatically finding a neural network architecture for a certain task such as CV and NPL (Chen et al., 2018; Liu et al., 2019a; Chen et al., 2019) and different datasets without human's labor of designing networks. Particularly, in real applications, the objective of NAS is more preferred to be obtaining a decent accuracy under a limited computational budget. Thus a multi-objective NAS is a more practical setting than only focusing on accuracy. There are several approaches in NAS area: 1) Reinforcement learning-based algorithm. Baker et al. (2016); Zoph & Le (2017); Cai et al. (2018) train an RNN policy controller to generate a sequence of actions to design cell structure for a specify CNN architecture; 2) Evolution-based algorithm. Real et al. (2017); Liu et al. (2018b); Real et al. (2019a) try to evolve architectures or Network Morphism by mutating the current best architectures and explore new potential models; 3) Gradient-based algorithm Liu et al. (2019b); Cai et al. (2019); Xie et al. (2019) define an architecture parameter for continuous relaxation of the discrete search space, thus allowing differentiable optimization of the architecture. Among those algorithms, most existing methods focus on a single objective (accuracy), others adding computation constraints as a regularization loss in the gradient-based method or as a reward in the RL-based algorithm. In contrast, our method reformulates the multi-objective NAS as a non-dominate sorting problem and further enables an efficient search over flexible customized search space.

## D Dataset and Search Space

### D.1 NasBench encoding

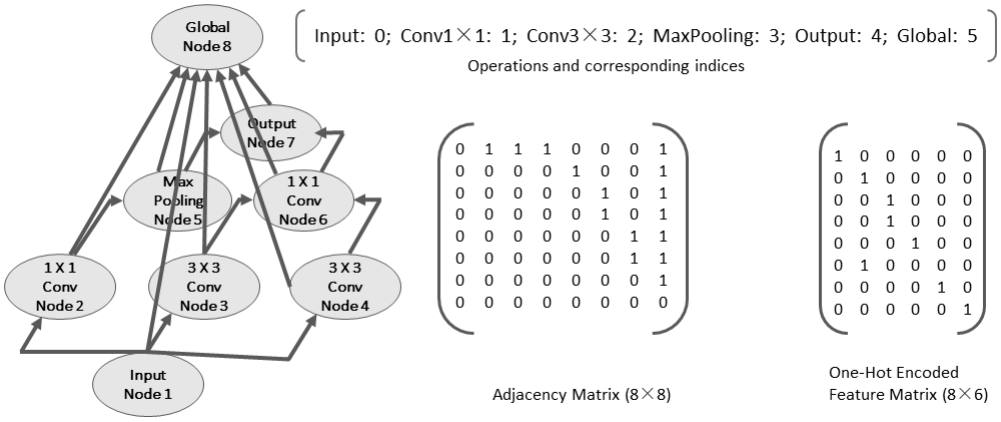



Figure 10: The encoding scheme of an example cell in NasBench.

### D.2 LSTM-12K DATASET

To create LSTM model dataset, we follow the same setting proposed by ENAS (Pham et al., 2018), we use adjacency matrix and a list of operators to represent an LSTM cell, and randomly sampled 12K cell structures from that domain, because the cells have natural graph structures, it is easy to feed them directly into GCN and conduct training. Due to the limitation on computational resources, we only sample architectures with number of nodes less than or equal to 8, and trained each cell for 10 epochs on PTB dataset (MARCUS et al., 1993). We use perplexity as a performance measure for the cells.

### D.3 RESNET STYLE SEARCH SPACE

We follow the setting of Li et al. (2019) to prepare the ResNet style search space. This search space aims to find when to perform down-sampling and when to double the channels. The ResNet style backbone consists of 5 stages with different resolutions from input images. The spatial size of Stage 1 to 5 is gradually down-sampled by the factor of 2. As suggested in Li et al. (2019), we fixed one $3 \times 3$ convolution layer ($stride = 2$) in Stage-1 and the beginning of Stage-2. We use the block setting as bottleneck residual block in ResNet. Then, the backbone architecture encoding string looks like "$1211 - 211 - 1111 - 12111$", where "$-$" separates each stage with different resolution, "1" means regular block with no change of channels and "2" indicated the number of base channels is doubled in this block. The base channel size is $64$. In Section 4.4, we just take "$-, 1, 2$" as three different operations and encode architectures as a series of strings. We train the model generated from this search space for $40$ epochs with a fast convergence learning rate schedule. Each architecture can be evaluated in $4$ hours on one server with $8$ V100 GPU machines.

## E BEST FOUND MODELS

Figure 11, 12 and 13 show the found architectures from the open domain search on ResNet style search space by our method.

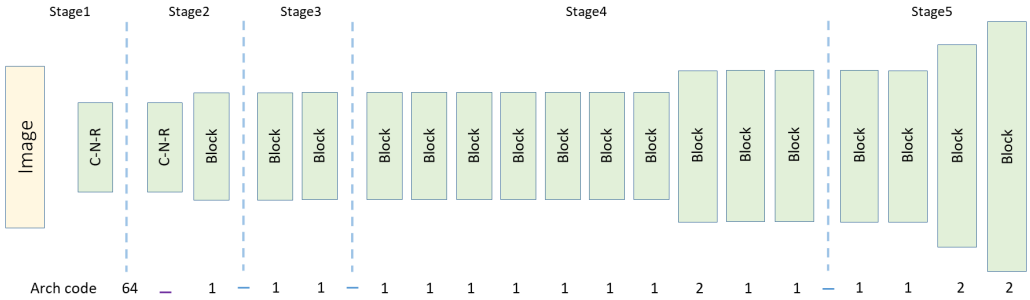



Figure 11: M1 ("$1 - 11 - 1111111211 - 1122$")

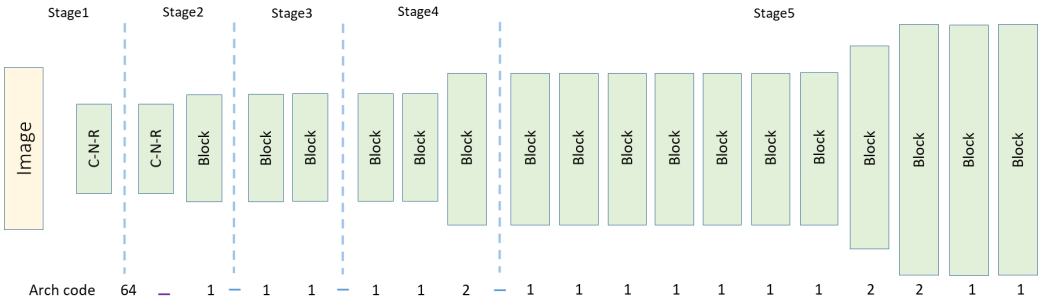



Figure 12: M2 ("1 − 11 − 112 − 11111112211")

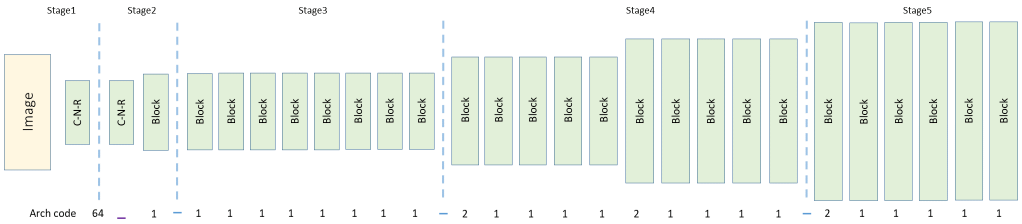



Figure 13: M3 ("1 − 11111111 − 2111121111 − 211111")