Efficient Sample-based Neural Architecture Search with Learnable Predictor

Han Shi [*1] **Renjie Pi** [*2] **Hang Xu** [2] **Zhenguo Li** [2] **James T. Kwok** [1] **Tong Zhang** [1]

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 NAS algorithm. For balancing exploitation and exploration, we propose BONAS (Bayesian Optimized Neural Architecture Search), a sample-based NAS framework combined with Bayesian Optimization. The main components of BONAS are Sampler and Learnable Embedding Extractor. Specifically, we apply Evolution Algorithm method as our sampler and apply Graph Convolutional Network predictor 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. Extensive experiments are conducted to verify the effectiveness of our method over many competing methods, e.g. $123.7\times$ more efficient than Random Search and $7.5\times$ more efficient than previous SOTA LaNAS for finding the best architecture on the largest NAS data set NAS-Bench-101.

1. Introduction

Designing an appropriate deep network architecture for each and every task / data set is tedious and time-consuming. Neural architecture search (NAS) (Zoph et al., 2018), which attempts to find this architecture automatically, has aroused significant recent interest. Results competitive with hand-crafted architectures have been obtained in many application areas, such as natural language processing (Luong et al., 2018; So et al., 2019) and computer vision (Real et al., 2019; Ghiasi et al., 2019; Chen et al., 2019; Liu et al., 2019a; Chu et al., 2019).

Optimization in NAS is difficult because the search space can contain billions of network architectures. Moreover, the objective value (e.g., accuracy) for a particular architecture is computationally expensive to evaluate. Hence, a central component in NAS is the strategy to search in such a huge space of architectures. These strategies can be broadly categorized into two groups. Sample-based algorithms (Zoph & Le, 2017; Liu et al., 2018; Real et al., 2019; Luo et al., 2018) sample candidate architectures with potentially high accuracies, and then evaluate them by full training. The second category contains one-shot NAS algorithms, which combine several architectures together using weight sharing (Pham et al., 2018) or continuous relaxation (Liu et al., 2019b; Luo et al., 2018) for faster training. However, due to the use of these approximations, they may not find top-performing models and can be sensitive to initialization (Sciuto et al., 2020; Yang et al., 2020). Note that even in close domain search, most one-shot NAS algorithms can only find an approximately optimal model.

Most NAS methods focus only on obtaining highly accurate models. However, the cost of searching for these competitive architectures can be expensive. In comparison, Bayesian Optimization (BO) is an efficient model for search and optimization problems, which explicitly considers exploitation and exploration (Mockus et al., 1978). In each iteration, the posterior distribution is updated with samples drawn from the search space by a cheap surrogate model.

While the Gaussian process (GP) is the most popular surrogate model for use with BO, its time complexity increases cubically with the number of samples (Snoek et al., 2015). Hence, GP is costly for use in NAS, as each sample corresponds to a candidate architecture and the architecture search space is huge. Another drawback for the use of GP in NAS is that it requires a kernel defined on architectures. Recently, (Kandasamy et al., 2018) and (Jin et al., 2019) provided two such heuristic distance formulations. However, they can be slow to compute on large architectures. It

[*]Equal contribution [1]The Hong Kong University of Science and Technology [2]Huawei Noahs Ark Lab. Correspondence to: Han Shi <hshiac@cse.ust.hk>.

Under review at ICML 2020

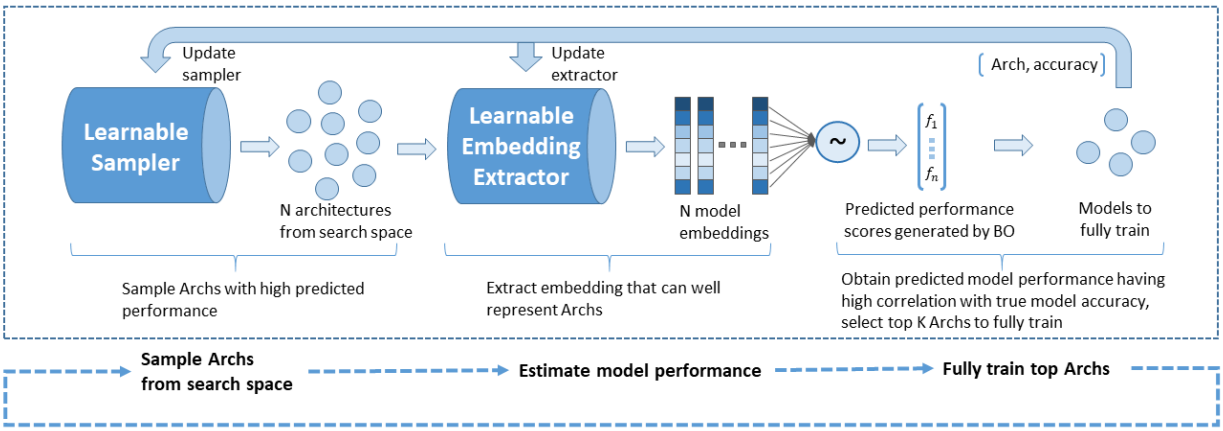


Figure 1. Overview of the proposed sample-based NAS algorithm using Bayesian Optimization. It has two main components: Learnable Embedding Extractor and Sampler. In every search iteration, we sample N architectures from the search space, estimate the model performance and fully train the model with top estimated performance.

is still an open problem on how to appropriately define an efficient neural architecture distance.

As for efficiently obtaining the performance of a particular architecture, a common approach is to perform prediction. Predictors based on the MLP and LSTM have been proposed (Liu et al., 2018; Luo et al., 2018; Wang et al., 2019b). However, they encode the architecture as a string, and ignore its connectivity structure. Besides, MLP can only operate on fixed-length inputs, and so cannot be used when the architectures have variable numbers of nodes.

To tackle the above two problems, we present BONAS (Bayesian Optimized Neural Architecture Search), which is a sample-based NAS algorithm using Bayesian Optimization. As shown in Figure 1, it has two main components: the Learnable Embedding Extractor and Sampler. In the Embedding Extractor, we use a graph convolutional network (GCN) to produce embeddings for neural architectures. This naturally handles the graph structures of neural architectures, and thus avoids the problems of the MLP and LSTM predictors. Together with Bayesian Linear Regression (BLR), this replaces the popular GP model as the surrogate function in BO. We found that the GCN can generalize well with a small number of architecture-accuracy training pairs. Graph neural network has been used in (Zhang et al., 2019) for the prediction of architecture parameters using a graph hypernetwork. However, it is a one-shot NAS method, and thus cannot ensure finding the optimal model even in close domain search. In contrast, we use graph embedding to predict the performance directly and can guarantee performance. The Sampler is used to sample architectures from the search space for exploration. In this paper, we will compare the commonly used samplers, including the random sampler, reinforcement learning based sampler and evolutionary algorithm based sampler.

Empirically, the proposed BONAS outperforms state-of-the-art methods including Evolution (Real et al., 2019), AlphaX (Wang et al., 2019b), and LaNAS (Wang et al., 2019a). We observe consistent gains on multiple search spaces for vision and NLP tasks, including the standard benchmark data sets of NAS-Bench-101 (Ying et al., 2019) and NAS-Bench-102 (Dong & Yang, 2020), and on LSTM-12K, a NAS benchmark data set we recently collected. In particular, BONAS is $123.7\times$ more efficient than random search, and $7.5\times$ more efficient than LaNAS on NAS-Bench-101. Our algorithm is also applied to open domain search with the NASNet search space (Zoph et al., 2018), and finds competitive models with fewer samples.

2. Related Work

2.1. Neural Architecture Search (NAS)

NAS tries to automatically find a suitable neural network architecture for a given task (Chen et al., 2018; Liu et al., 2019a; Chen et al., 2019). NAS algorithms can be classified into two categories. The first one includes one-shot NAS algorithms (Pham et al., 2018; Liu et al., 2019b; Luo et al., 2018), which only fully train the super-network once and then obtain the best sub-network. The second category includes sample-based NAS algorithms (Zoph & Le, 2017; Liu et al., 2018; Real et al., 2019; Luo et al., 2018), which sample a number of architectures and then fully train them.

Because of the approximations used in merging architectures to a super-network, one-shot methods may not guarantee performance of the obtained architecture (Sciuto et al., 2020; Yang et al., 2020). In this paper, we focus on sample-based methods, and the goal is to make them more efficient.

2.2. Bayesian Optimization

Bayesian Optimization (BO) can be used to find the globally optimal solution of an optimization problem (Mockus et al., 1978). Without loss of generality, here we consider maximizing an objective f. The Gaussian process (GP) is widely used as the underlying surrogate model for learning f (Jones, 2001).

In BO, an acquisition function is used to guide the search. Many acquisition functions have been proposed, with different ways to balance exploitation and exploration for the next proposed sampling (Shahriari et al., 2015). In particular, the *Expected Improvement* is often adopted (Mockus et al., 1978). Let the hyperparameters of the surrogate model be Θ. Given the observed data set $\mathcal{D}$ and new data x, define

$$\gamma(x) = \frac{\mu(x; \mathcal{D}, \Theta) - f(x_{best})}{\sigma(x; \mathcal{D}, \Theta)}, \tag{1}$$

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 maximum value of the objective observed so far. The *Expected Improvement* (EI) criterion is defined as:

$$\begin{aligned} &a_{EI}(x; \mathcal{D}, \Theta) \\ &= \sigma(x; \mathcal{D}, \Theta)[\gamma(x)\Phi(\gamma(x); 0, 1) + \mathcal{N}(\gamma(x); 0, 1)], \end{aligned} \tag{2}$$

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

2.3. Graph Convolutional Network

Given a graph $G = (V, E)$, where V is the set of N nodes, and E is the set of edges. Let the corresponding $N \times N$ adjacency matrix be A. Each node is associated with a d-dimensional feature, and the feature matrix for the whole graph is $X \in \mathbb{R}^{N \times d}$. The graph convolutional network (GCN) is a model for graph-structured data, which utilizes localized spectral filters to extract an useful embedding of each node (Kipf & Welling, 2016). It has been used on a variety of graph-related tasks, such as link prediction and clustering.

For a L-layer GCN, the layer-wise propagation rule is:

$$H^{(l+1)} = f(H^{(l)}, A) = \mathrm{ReLU}(\widetilde{D}^{\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}), \tag{3}$$

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. Proposed Method

To search for the optimal architecture more efficiently, we propose BONAS by using the Learnable Embedding Extractor (Section 3.1) while utilizing Bayesian Optimization (Section 3.2). To alleviate the problem introduced by the huge search space, we apply a Sampler to select a subspace for evaluation (Section 3.3). For NAS-oriented problems, we also propose a weighted loss function in Section 3.4. Figure 1 shows an overview of the proposed algorithm (Section 3.5).

As BO balances exploration and exploitation during search and the Learnable Embedding Extractor obtains embeddings that can well represent model architectures, the proposed algorithm is able to obtain top-performing architectures with fewer samples from the search space than the previous methods.

3.1. Learnable Embedding Extractor

A neural network can be represented as a directed attributed graph. Each node represents an operation (such as a 1×1 convolution in CNN, and ReLU activation in LSTM) and edges represent data flow (Zhang et al., 2019). Figure 2 shows an example on NAS-Bench-101. Since a NAS-Bench-101 architecture is obtained by stacking multiple repeated cells, we only consider the embedding of such a cell here. Thus, the NAS problem reduces to the search of cell architectures.

With this graph representation, it is natural to use a GCN for encoding. Compared to encoding using a MLP or LSTM as in (Liu et al., 2018; Luo et al., 2018; Wang et al., 2019b), the GCN can better preserve the graph's structural information (Kipf & Welling, 2016), and can easily handle graphs of various sizes. Specifically, graph connectivity is encoded by the adjacency matrix A, which can be obtained from the graph structure directly. Individual operations are encoded as one-hot vectors, and then aggregated to form the feature matrix X. Together, they are input to the GCN as in Section 2.3.

However, the standard GCN is only used to produce embeddings for nodes (Kipf & Welling, 2016), while here the target is to obtain an embedding for the whole graph. To solve this problem, following (Scarselli et al., 2008), we connect all nodes in the graph to an additional "global" node (Figure 2). The one-hot encoding scheme is also extended to include these new connections as a new operation. The embedding of the global node, obtained from (3), is then used as the embedding of the whole graph.

To train the GCN, we feed its output (cell embedding) to a standard regressor. In this paper, we use a single-hidden-layer neural network. Using the NAS-Bench data sets, the target output is the actual accuracy of the network constructed by stacking this particular cell. This regressor is then trained end-to-end with the GCN as in (Wang et al., 2019b).

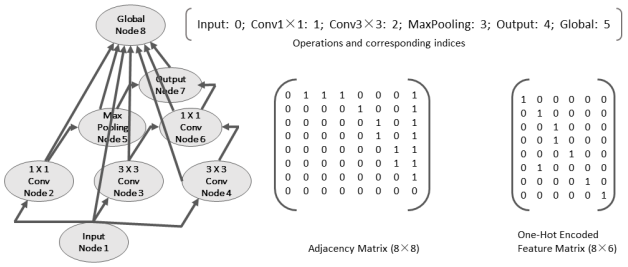


Figure 2. Encoding scheme of an example cell in NAS-Bench-101.

3.2. Bayesian Linear Regressor with Embedding Input

Recall that the GP is commonly used as the surrogate model in BO. For NAS problems, the GP requires defining a kernel on two networks (Kandasamy et al., 2018; Jin et al., 2019), which is often heuristic, difficult, and computationally expensive. Besides, training the GP model takes $\mathcal{O}(N^3)$ time, where N is the number of architectures sampled, and is again computationally expensive.

To alleviate these issues, inspired by (Snoek et al., 2015), we replace the GP by a Bayesian linear regressor (BLR) (Bishop, 2006) with the learned embedding in Section 3.1 as input. Let $\phi(A, X)$ be the embedding for an architecture (graph) with adjacency matrix A and feature matrix X. Given a set $\mathcal{D}$ of N training architectures $\{(A_i, X_i)\}$ with their actual performances $\{t_i\}$, the corresponding embeddings $\{\phi(A_i, X_i)\}$ are concatenated to form the design matrix Φ, with $\Phi_{ij} = \phi_j(A_i, X_i)$. For a new (A, X), the predicted mean μ from the BLR model is given by (Bishop, 2006):

$$\mu(A, X; \mathcal{D}, \alpha, \beta) = m_N^T \phi(A, X), \tag{4}$$

where $m_N = \beta S_N \Phi^T t$, $S_N = (\alpha I + \beta \Phi^T \Phi)^{-1}$, I is the identity matrix, and (α, β) are precision parameters of the model that can be estimated by maximizing the marginal likelihood as follows(Snoek et al., 2012).

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

By considering only the uncertainty of the weights in the last fully-connected layer, the prediction variance is obtained as (Snoek et al., 2015):

$$\sigma^2(A, X; \mathcal{D}, \alpha, \beta) = \phi(A, X)^T S_N \phi(A, X) + 1/\beta. \tag{5}$$

From the obtained μ and σ^2, one can compute the expected improvement score from (2), and then use this to select the next (architecture) sample.

3.3. Sampling the Search Space

Since the NAS search space is huge, only a subset can be evaluated in each BO iteration. A standard sampling approach is random search (Yang et al., 2020), which samples architectures in the search space randomly. Another popular approach is reinforcement learning (RL). Here, unlike (Zoph et al., 2018) which uses the accuracy of the fully trained network as reward for controller training, we use the EI score as reward. The other training settings are the same. Since computing the EI score is much less expensive than full training a model, we can sample more architectures with their EI scores for RL controller training.

We also experiment with a recent evolutionary algorithm (EA) in (Real et al., 2019), which uses aging evolution and tournament selection. New architectures are constructed by mutation, which can be performed on either the adjacency matrix[1] or feature matrix. To mutate the adjacency matrix, we randomly select one edge $i \to j$ in the cell, and modify its source i to a random node preceding i in the computation graph. To mutate the feature matrix, we randomly select a node in the cell, and change its operation to any possible operation (including the original one) with equal probabilities. Among these three samplers, the RL-based sampler is learnable while the other two are not.

3.4. Exponential Weighted Loss

In Section 3.1, we train the GCN with the MSE loss. However, on training the surrogate function for exploration and exploitation (Section 3.2), more attention should be paid to the high-performing architectures. Specifically, recall that in BO, we select samples with the largest acquisition function value. Hence, it is desirable to predict architectures with high performance as accurately as possible so that the best one can be found, while predictions for low-performing architectures can be less accurate.

To achieve this, we propose the following exponential weighted loss for training the surrogate predictor model during the search phase of BO:

$$L_{exp} = \frac{1}{N(e-1)} \sum_{i=1}^{N} (\exp(\widetilde{y_i}) - 1)||y_i - \widetilde{y_i}||^2. \tag{6}$$

Here, y_i is the predicted accuracy[2], $\widetilde{y_i}$ is the ground-truth accuracy, and $e-1$ is the normalization factor (note that e is the base of the natural logarithm). Thus, the loss will have more emphasis on models with higher accuracies. Note that our idea is inspired by the focal loss (Lin et al., 2017), but the focal loss is used for classification while ours is used for

[1] On NAS-Bench-102 (Dong & Yang, 2020), the adjacency matrix is fixed and not mutated.

[2] This can be easily adapted when metrics other than accuracy are used.

Algorithm 1 Bayesian Optimized Neural Architecture Search (BONAS). $\mathcal{A}$ is the given search space, N is the number of initial architectures, k is the ratio of GCN / BLR update times.

initialize random N fully-trained architectures: $\mathcal{D} = \{(A_i, X_i, t_i)\}_{i=1}^{N}$ from search space $\mathcal{A}$;
initial training of GCN using $\mathcal{D}$ with proposed loss;
replace the final layer of GCN with BLR;
initialize Sampler;
repeat
  for iteration$= 1, 2, \ldots, k$ **do**
    sample candidate pool $\mathcal{C}$ from $\mathcal{A}$;
    for each candidate m in $\mathcal{C}$ **do**
      embed m using GCN;
      compute μ and σ^2 in (4) and (5) using BLR;
      compute expected improvement (EI) in (2);
    end for
    $M \leftarrow$ candidate with the highest EI score;
    fully train M to obtain its actual performance;
    add M and its actual performance to $\mathcal{D}$;
    update BLR using the enlarged $\mathcal{D}$;
    update Sampler;
  end for
  retrain GCN using the enlarged $\mathcal{D}$ with proposed loss;
until stop criterion satisfy.

regression.

3.5. Algorithm

The whole procedure, which will be called BONAS (Bayesian Optimized Neural Architecture Search), is shown in Algorithm 1. Given the search space $\mathcal{A}$, we start with a set $\mathcal{D}$ of N random architectures $\{(A_i, X_i)\}$, which have been trained and the corresponding performance values $\{t_i\}$ known. The GCN embedding extractor is then trained using $\mathcal{D}$ (section 3.1). Afterwards, the last fully-connected layer is replaced by BLR (Section 3.2).

In each search iteration, a pool $\mathcal{C}$ of candidates are sampled from $\mathcal{A}$ by Sampler. For each candidate, its expected improvement score is computed from Eq. (2) by using its GCN embedding and predicted mean / variance from BLR. The candidate with the highest EI score is selected and fully trained to obtain its actual performance value. This is added to $\mathcal{D}$, and then the GCN predictor and BLR are updated. As GCN training is more expensive than BLR updating, the BLR is updated more frequently than the GCN. For Sampler updating, we update the population set by adding the new model and removing the old model for EA-based sampler. For RL-based sampler, we sample R architectures using RL controller, obtain their EI score as reward to update the controller.

The procedure is repeated until satisfying the stop criteria. For example, the stop criteria can be achieving a certain number of fully trained architectures, or achieving the best architecture in the search space (for close search space only).

4. Experiments

In the following experiments, we will use NAS-Bench-101 (Ying et al., 2019), which is the largest benchmark NAS data set with 423K convolutional architectures; and the more recent NAS-Bench-102 (Dong & Yang, 2020), which uses a different search space (with 15K architectures) and is applicable to almost any NAS algorithm. As both NAS-Bench-101 and NAS-Bench-102 focus on convolutional architectures, we also construct another benchmark data set[3] (denoted LSTM-12K), containing 12K LSTM models trained on the Penn TreeBank data set (Marcus et al., 1993). Details are in Appendix A. Experiments are also performed in the open domain scenario using the NASNet search space (Zoph et al., 2018). All experiments are performed on the NVIDIA Tesla V100 GPUs.

4.1. Predictor Comparison

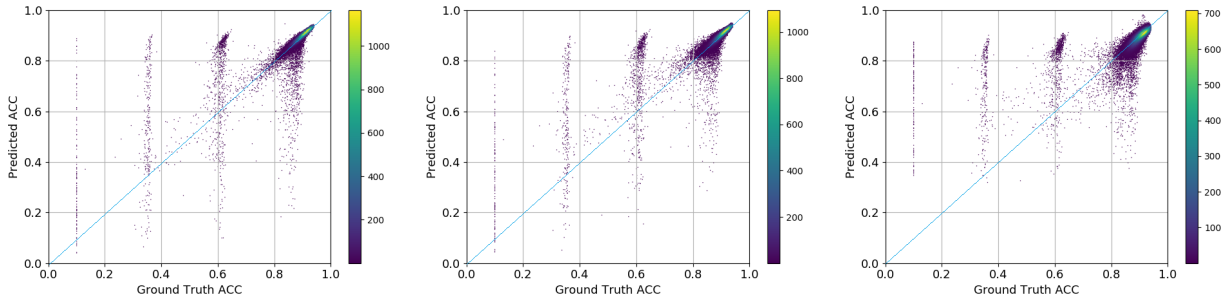
In this section, we first demonstrate superiority of the proposed GCN predictor over existing MLP and LSTM predictors in (Wang et al., 2019b). The GCN has four hidden layers with 64 units each. Training is performed by minimizing the square loss, using the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 0.001 and a mini-batch size of 128. As in (Wang et al., 2019b), the MLP predictor has 5 fully-connected layers, with 512, 2048, 2048, 512 and 1 units, respectively. As for the LSTM, the sizes of both the hidden layer and embedding are 100. The last LSTM hidden layer is connected to a fully-connected layer.

Experiments are performed on the NAS-Bench-101, NAS-Bench-102, and LSTM-12K data sets. For each data set, we use 85% of the data for training, 10% for validation, and the rest for testing. For performance evaluation, as in (Wang et al., 2019b), we use the correlation coefficient between the model's predicted and actual performance values (testing accuracy on NAS-Bench-101 and NAS-Bench-102, and perplexity on LSTM-12K).

Table 1. Correlation (%) between the model's predicted and actual accuracies.

	NAS-Bench-101	NAS-Bench-102	LSTM-12K
MLP	83.0	86.5	53.0
LSTM	74.1	79.5	56.0
GCN	**84.1**	**97.3**	**74.2**

[3]This data set will be be released after paper publication.



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

Figure 3. Predicted accuracies by various predictors vs the actual accuracies on NAS-Bench-101. The value shown in the color bar is the pdf value from a Gaussian kernel density estimator fitted on the points.

Table 1 shows the correlation results. The predicted and actual accuracies on NAS-Bench-101 are also illustrated in Figure 3 (plots for NAS-Bench-102 and LSTM-12K are in Appendix B). As can be seen, the GCN predicts the performance more accurately than the other two predictors.

4.2. Closed Domain Search

In this section, we study the efficiency of the proposed BONAS algorithm for neural architecture search on the NAS-Bench-101, NAS-Bench-102 and LSTM-12K data sets. In step 1 of Algorithm 1, we start with 10 random architectures that are fully trained (these 10 are counted towards the total number of architectures sampled). The candidate pool C is of size 1,000, and k in step 4 is 10. For RL-based sampler, the number of training set for controller updating R is 1000.

We compare BONAS with the following state-of-the-art sample-based NAS baselines: (i) Random search (Yang et al., 2020), which explores the search space randomly without exploitation; (ii) Regularized evolution (Real et al., 2019), which uses an heuristic evolution process for exploitation; (iii) NASBOT (Kandasamy et al., 2018), which uses BO with a manually-defined kernel on architectures; (iv) Neural Architecture Optimization (NAO) (Luo et al., 2018), which finds the architecture in a continuous embedding space with gradient descent; (v) AlphaX (Wang et al., 2019b); and (vi) LaNAS (Wang et al., 2019a). Both AlphaX 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 candidate pool. The experiment is repeated 50 times and the average results reported.

Table 2 shows the number of architectures each method samples before the optimal architecture is found. As can be seen, the proposed BONAS consistently outperforms the other baselines. On NAS-Bench-101, BONAS(EA) is $123.7\times$, $57.5\times$, $48.6\times$, $36.7\times$, $39.8\times$, $7.5\times$ more sample-efficient than random search, regularized evolution, AlphaX, NAO, NASBOT and LaNAS, respectively. On the smaller NAS-Bench-102 and LSTM-12K data sets, BONAS can still find the optimal architecture with fewer samples.

Among the three search space sampling schemes, we empirically found that the EA-based sampler is more sample-efficient than the other two. In the sequel, the EA-based sampler and GCN predictor will be the default setting of BONAS.

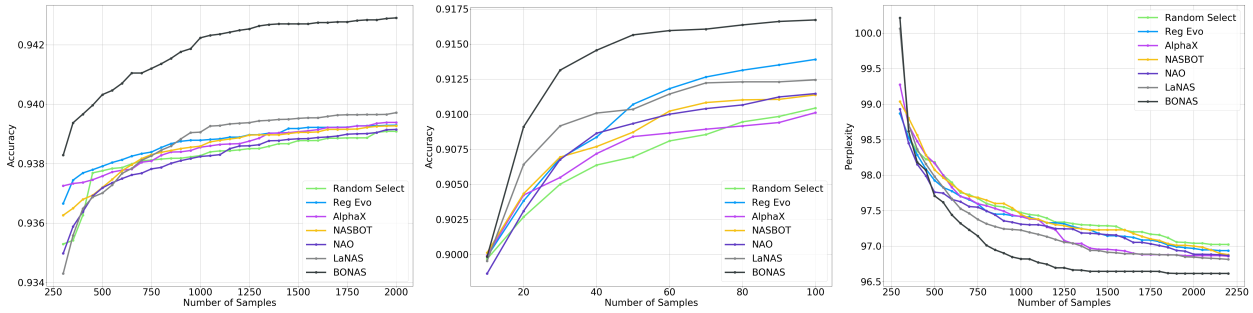
Table 2. Number of architectures sampled before the best architecture in the closed domain is found (averaged over 50 repetitions). Here, BONAS(RS), BONAS(RL), and BONAS(EA) denote the BONAS variants with random sampling, reinforcement learning and evolutionary algorithm as sampling scheme, respectively.

	NAS-Bench-101	NAS-Bench-102	LSTM-12K
Random	188139.8	7712.3	6182.6
Reg Evolution	87402.7	535.5	5670.9
NASBOT	60589.5	1549.3	4092.8
NAO	55828.8	1589.2	3845.6
AlphaX	73977.2	1639.1	4687.4
LaNAS	11390.7	1367.4	2464.4
BONAS(RS)	3587.8	320.0	2388.4
BONAS(RL)	2804.5	308.4	1080.6
BONAS(EA)	**1520.9**	**72.5**	**697.6**

Following (Wang et al., 2019a;b), we also report the performance of the best model in each search iteration. As shown in Figure 4, the proposed BONAS outperforms other search algorithms in the number of samples consistently.

4.3. Open Domain Search

In this section, we perform NAS in the open domain, using the NASNet search space (Zoph et al., 2018) on the CIFAR-10 data set. Following (Liu et al., 2019b), we allow



(a) NAS-Bench-101. (b) NAS-Bench-102. (c) LSTM-12K.

Figure 4. Performance of the best model in each search iteration.

Table 3. Comparison of different methods on CIFAR-10. Blocks is the number of the blocks inside the cell (Zoph et al., 2018) and cutout is an additional enhancement (DeVries & Taylor, 2017). "-" in No. of samples truly evaluated means one-shot NAS method.

Model	Blocks	Params	Top-1 err (%)	No. of samples truly evaluated	GPU days
NASNet-A+cutout (Zoph et al., 2018)	5	3.3 M	2.65	20000	2000
AmoebaNet-B+cutout (Real et al., 2019)	5	2.8 M	2.55	27000	3150
PNASNet-5 (Liu et al., 2018)	4	3.2 M	3.41	1160	225
NAO (Luo et al., 2018)	5	10.6 M	3.18	1000	200
LaNet (Wang et al., 2019a)	7	3.2 M	2.53	803	150
ENAS+cutout (Pham et al., 2018)	5	4.6 M	2.89	-	0.45
DARTS+cutout (Liu et al., 2019b)	4	3.3 M	2.76	-	1.5
BayesNAS+cutout (Zhou et al., 2019)	4	3.4 M	2.81	-	0.2
ASNG-NAS+cutout (Akimoto et al., 2019)	5	3.9 M	2.83	-	0.11
BONAS+cutout	4	3.5M	**2.44**	**186**	**120**

4 blocks inside a cell. To save training time, in step 12 of Algorithm 1, instead of performing full training, we stop training of each sampled architecture after 120 epochs. The other experimental settings are the same as in Section 4.2.

The proposed BONAS algorithm is compared with SOTA sample-based NAS algorithms: (i) NASNet, which uses reinforcement learning to sample architecture directly (Zoph et al., 2018); (ii) AmoebaNet, which finds the architecture by evolution algorithm (Real et al., 2019) ; (iii)PNASNet, which searches architecture progressively combined with a predictor (Liu et al., 2018); (iv) NAO (Luo et al., 2018); (v) LaNet, which learns action space by Monte Carlo Tree (Wang et al., 2019a). For comparison, we also illustrate the results of some one-shot NAS methods: (i) ENAS, which finds the model by parameter sharing (Pham et al., 2018); (ii) DARTS, which applies continuous relaxation for super-net training (Liu et al., 2019b); (iii) BayesNAS, which considers the dependency in architecture (Zhou et al., 2019); (iv) ASNG-NAS, which proposes a stochastic natural gradient method for NAS problem (Akimoto et al., 2019).

As the optimal architecture is not known in open domain search, we select the best performing architecture M at 186 samples and fully train it. Table 3 compares our result with other baselines on CIFAR-10. As can be seen, compared with the other sample-based NAS methods, BONAS outperforms all methods with fewer number of evaluated samples and fewer GPU days. Specifically, even though we use $4.32\times$ fewer samples than previous SOTA LaNAS, we can still find a better architecture. Although the time cost of one-shot NAS algorithms is much less than BONAS, the top-1 error of the final found model is higher. The structure of the best architecture achieved by BONAS is in Appendix C.

Note that during the search phase, different choices of training epochs may lead to different results, as architectures with fewer parameters tend to converge faster. Architectures with many skip-connects are selected if the number of training epochs is too small. We need to find a balance between time cost and fidelity of model accuracy.

4.4. Transfer Learning

For transferability study, we consider two scenarios: data set transfer and search space transfer. For data set transfer, we test the architecture learned on CIFAR-10 and transferred to ImageNet (Section 4.4.1). For search space transfer, we consider two different search spaces and compare the proposed method with pre-training (Section 4.4.2).

Table 4. Comparison of different methods on ImageNet (with mobile setting). Here, Blocks is the number of the blocks inside the cell (Zoph et al., 2018), and Mult-Adds is the number of multiply-add operations.

Model	Blocks	Mult-Adds	Params	Top1 / Top5 err (%)
NASNet-A	5	564 M	5.3 M	26.0 / 8.4
NASNet-B	5	488 M	5.3 M	27.2 / 8.7
NASNet-C	5	558 M	4.9 M	27.5 / 9.0
AmoebaNet-A	5	555 M	5.1 M	25.5 / 8.0
AmoebaNet-B	5	555 M	5.3 M	26.0 / 8.5
AmoebaNet-C	5	570 M	6.4 M	24.3 / 7.6
PNASNet-5	4	588 M	5.1 M	25.8 / 8.1
DARTS	4	574 M	4.7 M	26.7 / 8.7
BayesNAS	4	-	3.9 M	26.5 / 8.9
LaNet	7	570 M	5.1 M	25.0 / 7.7
BONAS	4	561 M	5.1 M	25.6 / 8.1

4.4.1. Data Set Transfer

As in (Liu et al., 2019b; Luo et al., 2018; Wang et al., 2019a), we test the performance of the best found architecture on CIFAR-10 when transferred to ImageNet (Deng et al., 2009). For fair comparison, we follow the mobile setting (Zoph et al., 2018; Liu et al., 2019b), in which the size of input image is 224×224 and the number of multiply-add operations is constrained to be fewer than 600M. The training setup is the same as (Liu et al., 2019b). As shown in Table 4, in the same search space with 4 blocks, the transferred architecture found by BONAS outperforms the others consistently. Even though compared with all the baselines (different blocks), the transferred architecture can still achieve competitive results.

4.4.2. Search Space Transfer

In this section, we study the ability of the proposed algorithm on search space transfer. Most NAS algorithms only focus on a static search space. In contrast, how to adapt the methods for extension of search space is still an open problem. For instance, after searching in a small search space $\mathcal{A}_1$, how to transfer the obtained knowledge to a larger search space $\mathcal{A}_2$. In this experiment, we split NAS-Bench-101 into two subsets: architectures with 6 nodes (62K) and architectures with 7 nodes (359K). Using the same settings as in Section 4.2, we pre-train our GCN model on architectures with 6 nodes, and then transfer the GCN predictor to search in the architecture space with 7 nodes. For comparison, we run the same algorithm without pretraining.

The search method with the pretrained GCN predictor finds the optimal model after 511.9 samples (averaged over 50 rounds), while the method without pretraining needs 1386.4 samples. As can be seen, pretraining can find the optimal model $2.7\times$ more efficiently than the unpretrained method, validating the transfer ability of the GCN predictor. Thus, BONAS can handle architectures of different scales as long as their operation choices are the same.

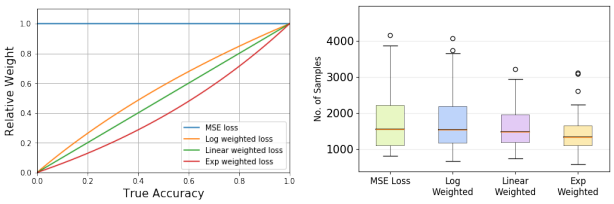
4.5. Ablation Study

To demonstrate the improvement of the proposed weighted loss, we compare it with MSE loss. Besides that, we also design two other weighted losses as following to show the influence of the second-order derivative of the added weight:

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

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

We use the same settings as in Section 4.2 but apply different loss functions for algorithm performing. Experimental result on NAS-Bench-101 is shown in Figure 5. As can be seen, the use of exponential weighted loss outperforms the other three losses. Even though we use traditional MSE loss for predictor training, our framework can still achieve the best architecture with fewer samples. It can be found that our method can reach the highest efficiency with exponential weighted loss.



(a) the four losses considered. (b) performance of four losses over 50 rounds.

Figure 5. Ablation study of Weighted Loss.

5. Conclusion

In this work, we propose BONAS, a sample-based NAS method combined with Bayesian Optimization. Instead of using popular Gaussian Processes surrogate model, we replace it with a learnable GCN predictor. For feasible evaluation, we apply a EA-based sampler to obtain a subspace every iteration. We utilize the efficiency of BO to search top-performance architectures. For NAS-specific tasks, we also propose a weighted loss focusing on top-performance models. Compared with other sample-based NAS algorithms, experimental results show that the proposed algorithm outperforms the SOTA LaNAS on NAS-Bench-101, NAS-Bench-102, LSTM-12K searching. As for open domain search, we also validate BONAS in NASNet search space, the found architecture can achieve 2.44% Top-1 error on CIFAR-10 with only 186 samples.

References

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

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

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

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

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

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

DeVries, T. and Taylor, G. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Dong, X. and Yang, Y. Nas-bench-102: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

Ghiasi, G., Lin, T., and Le, Q. 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.

Jin, H., Song, Q., and Hu, X. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

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

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

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

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

Li, X., Zhou, Y., Pan, Z., and Feng, J. 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.

Lin, T., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

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

Liu, C., Chen, L., Schroff, F., Adam, H., Hua, W., Yuille, A., and Fei-Fei, L. 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.

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

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

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

Marcus, M., Santorini, B., and Marcinkiewicz, M. Building a large annotated corpus of english: the penn treebank. *Computational linguistics-Association for Computational Linguistics*, pp. 313–330, 1993.

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

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

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

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

Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020.

Shahriari, B., Swersky, K., Wang, Z., Adams, R., and De Freitas, N. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 2015.

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

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

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

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

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

Yang, A., Esperança, P., and Carlucci, F. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020.

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

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

Zhou, H., Yang, M., Wang, J., and Pan, W. Bayesnas: A bayesian approach for neural architecture search. In *International Conference on Machine Learning*, 2019.

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

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

Appendix for: Efficient Sample-based Neural Architecture Search with Learnable Predictor

A. Data set

A.1. LSTM-12K data set

To create LSTM model data set, 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. In the defined search space, 4 activation functions (tanh, ReLU, identity, and sigmoid) are allowed and the architectures are decided by the activation function options and node connection. Due to the limitation on computational resources, we only sample architectures with number of nodes less than or equal to 8. We randomly sampled 12K cell structures from that domain, and trained each cell for 10 epochs on PTB data set (Marcus et al., 1993). Other training setups are the same with (Pham et al., 2018). We use perplexity as a performance measure for the cells.

A.2. NASNet search space

We follow the search space setting of DARTS (Liu et al., 2019b), in which the architecture is stacked by the learned cell. The cell consists of 4 blocks, two inputs (the output of previous cell and the previous previous cell) and one output. There are 8 types operations allowed: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zero. Similar to previous work (Liu et al., 2018), we apply the same cell architecture for both "normal" and "reduction" layer. For adapting to proposed GCN predictor, we regard the operation as the node and regard the data flow as the edge.

B. Predictor Comparison

Figure 6 and Figure 7 are the visualization of predictor performance on NAS-Bench-102 and LSTM-12K, respectively.

C. Best Found Models

Figure 8 shows the found architectures from the open domain search on NASNet search space by our method.

D. ResNet Style Search

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 3×3 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 this section, 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.

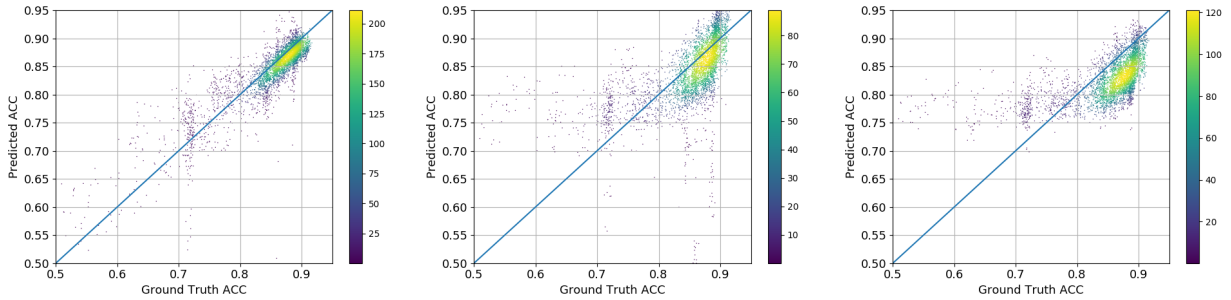
We validate the proposed BONAS for accuracy search on ImageNet (Deng et al., 2009). Due to the large volume of the dataset, we train the sampled architectures by early stopping at 40 epochs rather than fully training. And other experiment settings are the same with Section 4.3 as well.

With 500 samples, We fully train the most potential model so far. The comparison of our found models and famous ResNet family models are shown in Table 5. It shows that the model found by BONAS can achieve better accuracy performance compared with famous ResNets.

Figure 9 shows the found architecture from the open domain search on ResNet style search space by our method.

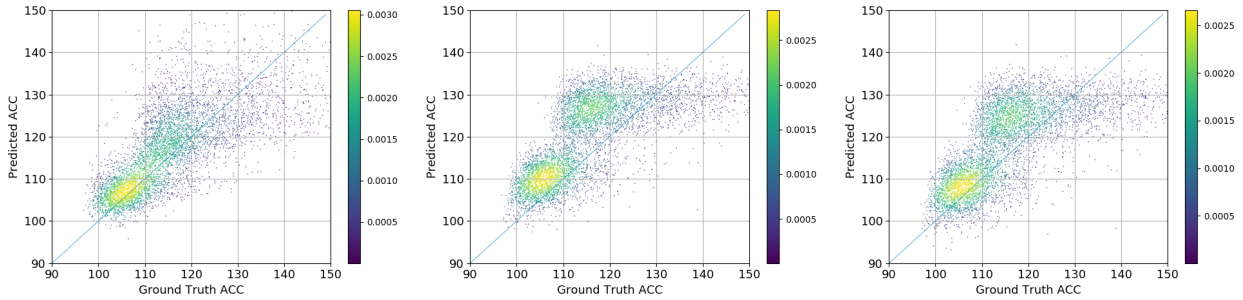
Table 5. Comparison of different ResNet style models on ImageNet.

Model	Params	Top1 error (%)
ResNet-18	11.68 M	30.4
ResNet-34	21.8 M	26.7
ResNet-50	25.56 M	24.0
ResNet-101	44.55 M	22.4
BONAS	36.56 M	21.7



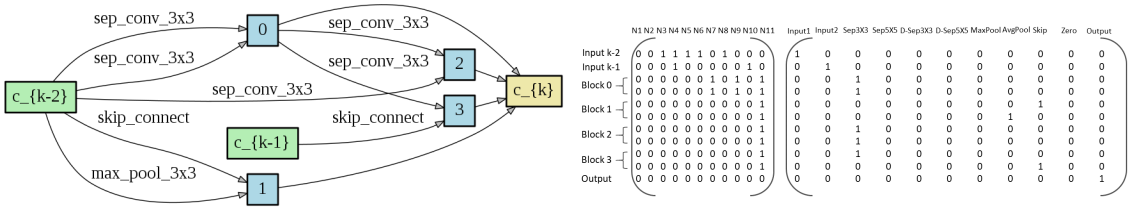
(a) GCN. (b) MLP. (c) LSTM.

Figure 6. Predicted accuracies by various predictors vs the actual accuracies on NAS-Bench-102. The value shown in the color bar is the pdf value from a Gaussian kernel density estimator fitted on the points.



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

Figure 7. Predicted accuracies by various predictors vs the actual accuracies on LSTM-12K. The value shown in the color bar is the pdf value from a Gaussian kernel density estimator fitted on the points.



(a) Normal cell and reduction cell (b) Cell encoding

Figure 8. The found models in NASNet search space.

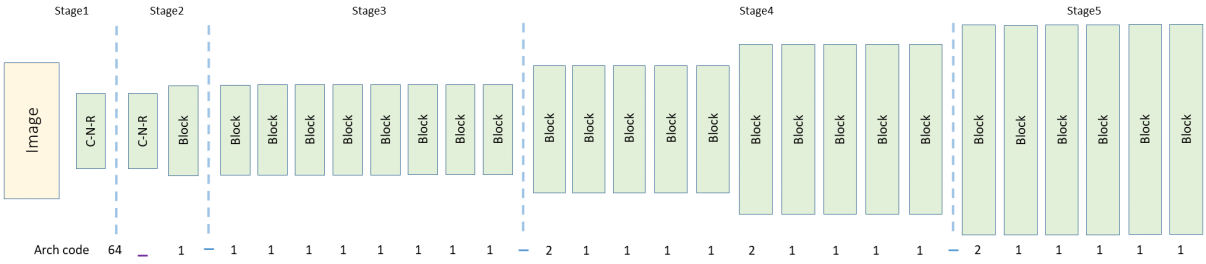


Figure 9. Model("1 − 11111111 − 2111121111 − 211111")