BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search

Colin White,[1] **Willie Neiswanger,** [2 3] **Yash Savani** [1]
[1] Abacus.AI
[2] Stanford University
[3] Petuum, Inc.
colin@abacus.ai, neiswanger@cs.stanford.edu, yash@abacus.ai

Abstract

Over the past half-decade, many methods have been considered for neural architecture search (NAS). Bayesian optimization (BO), which has long had success in hyperparameter optimization, has recently emerged as a very promising strategy for NAS when it is coupled with a neural predictor. Recent work has proposed different instantiations of this framework, for example, using Bayesian neural networks or graph convolutional networks as the predictive model within BO. However, the analyses in these papers often focus on the full-fledged NAS algorithm, so it is difficult to tell which individual components of the framework lead to the best performance.

In this work, we give a thorough analysis of the "BO + neural predictor" framework by identifying five main components: the architecture encoding, neural predictor, uncertainty calibration method, acquisition function, and acquisition function optimization. We test several different methods for each component and also develop a novel path-based encoding scheme for neural architectures, which we show theoretically and empirically scales better than other encodings. Using all of our analyses, we develop a final algorithm called BANANAS, which achieves state-of-the-art performance on NAS search spaces. We adhere to the NAS research checklist (Lindauer and Hutter 2019) to facilitate best practices, and our code is available at https://github.com/naszilla/naszilla.[1]

Introduction

Since the deep learning revolution in 2012, neural networks have been growing increasingly more complex and specialized (Krizhevsky, Sutskever, and Hinton 2012; Huang et al. 2017; Szegedy et al. 2017). Developing new state-of-the-art architectures often takes a vast amount of engineering and domain knowledge. A rapidly developing area of research, neural architecture search (NAS), seeks to automate this process. Since the popular work by Zoph and Le (2017), there has been a flurry of research on NAS (Liu et al. 2018; Pham et al. 2018; Liu, Simonyan, and Yang 2018; Kandasamy et al. 2018b; Elsken, Metzen, and Hutter 2018; Jin, Song, and Hu 2018). Many methods have been proposed, including evolutionary search, reinforcement learning, Bayesian optimization (BO), and gradient descent. In certain settings, zeroth-order (non-differentiable) algorithms such as BO are of particular interest over first-order (one-shot) techniques, due to advantages such as simple parallelism, joint optimization with other hyperparameters, easy implementation, portability to diverse architecture spaces, and optimization of other/multiple non-differentiable objectives.

BO with Gaussian processes (GPs) has had success in deep learning hyperparameter optimization (Golovin et al. 2017; Falkner, Klein, and Hutter 2018), and is a leading method for efficient zeroth order optimization of expensive-to-evaluate functions in Euclidean spaces. However, initial approaches for applying GP-based BO to NAS came with challenges that limited its ability to achieve state-of-the-art results. For example, initial approaches required specifying a distance function between architectures, which involved cumbersome hyperparameter tuning (Kandasamy et al. 2018b; Jin, Song, and Hu 2018), and required a time-consuming matrix inversion step.

Recently, Bayesian optimization with a neural predictor has emerged as a high-performing framework for NAS. This is similar to GP-based BO, except the GP is replaced with a neural network that is trained to predict the accuracy of unseen neural networks. This framework avoids the aforementioned problems with BO in NAS: there is no need to construct a distance function between architectures, and the neural predictor scales far better than a GP model. Recent work has proposed different instantiations of this framework, for example, Bayesian neural networks with BO (Springenberg et al. 2016), and graph neural networks with BO (Shi et al. 2019; Ma, Cui, and Yang 2019). However, the analyses often focus on the full-fledged NAS algorithm, making it challenging to tell which components of the framework lead to the best performance.

In this work, we start by performing a thorough analysis of the "BO + neural predictor" framework. We identify five major components of the framework: architecture encoding, neural predictor, uncertainty calibration method, acquisition function, and acquisition function optimization. For example, graph convolutional networks, variational autoencoder-based networks, or feedforward networks can be used for the neural predictor, and Bayesian neural networks or different types of ensembling methods can be used for the uncertainty calibration method. After conducting experiments on all components of the BO + neural predictor framework, we use this analysis to define a high-performance instantiation of the framework,

[1]Full-length paper here: https://arxiv.org/abs/1910.11858.

which we call BANANAS: Bayesian optimization with neural architectures for NAS.

In order for the neural predictor to achieve the highest accuracy, we also define a novel path-based architecture encoding, which we call the path encoding. The motivation for the path encoding is as follows. Each architecture in the search space can be represented as a labeled directed acyclic graph (DAG) – a set of nodes and directed edges, together with a list of the operations that each node (or edge) represents. However, the adjacency matrix can be difficult for the neural network to interpret (Zhou et al. 2018), since the features are highly dependent on one another. By contrast, each feature in our path encoding scheme represents a unique path that the tensor can take from the input layer to the output layer of the architecture. We show theoretically and experimentally that this encoding scales better than the adjacency matrix encoding, and allows neural predictors to achieve higher accuracy.

We compare BANANAS to a host of popular NAS algorithms including random search (Li and Talwalkar 2019), DARTS (Liu, Simonyan, and Yang 2018), regularized evolution (Real et al. 2019), BOHB (Falkner, Klein, and Hutter 2018), NASBOT (Kandasamy et al. 2018b), local search (White, Nolen, and Savani 2020), TPE (Bergstra et al. 2011), BONAS (Shi et al. 2019), BOHAMIANN (Springenberg et al. 2016), REINFORCE (Williams 1992), GP-based BO (Snoek, Larochelle, and Adams 2012), AlphaX (Wang et al. 2018), ASHA (Li and Talwalkar 2019), GCN Predictor (Wen et al. 2019), and DNGO (Snoek et al. 2015). BANANAS achieves state-of-the-art performance on NASBench-101 and is competitive on all NASBench-201 datasets. Subsequent work has also shown that BANANAS is competitive on NASBench-301 (Siems et al. 2020), even when compared to first-order methods such as DARTS (Liu, Simonyan, and Yang 2018), PC-DARTS (Xu et al. 2019), and GDAS (Dong and Yang 2019).

Finally, to promote reproducibility, in the full version of the paper we discuss how our experiments adhere to the NAS best practices checklist (Lindauer and Hutter 2019). In particular, we experiment on well-known search spaces and NAS pipelines, run enough trials to reach statistical significance, and release our code.

Our contributions. We summarize our main contributions.

- We analyze a simple framework for NAS: Bayesian optimization with a neural predictor, and we thoroughly test five components: the encoding, neural predictor, calibration, acquisition function, and acquisition function optimization.
- We propose a novel path-based encoding, which improves the accuracy of neural predictors. We give theoretical and experimental results showing that the path encoding scales better than the adjacency matrix encoding.
- We use our analyses to develop BANANAS, a high performance instantiation of the above framework. We empirically show that BANANAS is state-of-the-art on popular NAS benchmarks.

Related Work

NAS has been studied since at least the 1990s and has gained significant attention in the past few years (Kitano 1990; Stanley and Miikkulainen 2002; Zoph and Le 2017). Some of the most popular recent techniques for NAS include evolutionary algorithms (Maziarz et al. 2018), reinforcement learning (Zoph and Le 2017; Pham et al. 2018), BO (Kandasamy et al. 2018b), and gradient descent (Liu, Simonyan, and Yang 2018). For a survey of neural architecture search, see (Elsken, Metzen, and Hutter 2018).

Initial BO approaches defined a distance function between architectures (Kandasamy et al. 2018b; Jin, Song, and Hu 2018). There are several works that predict the validation accuracy of neural networks (Klein et al. 2017; Deng, Yan, and Lin 2017; Istrate et al. 2019; Zhang, Ren, and Urtasun 2018; Baker et al. 2017). A few recent papers have used Bayesian optimization with a graph neural network as a predictor (Ma, Cui, and Yang 2019; Shi et al. 2019), however, they do not conduct an ablation study of all components of the framework. In this work, we do not claim to invent the BO + neural predictor framework, however, we give the most in-depth analysis that we are aware of, which we use to design a high-performance instantiation of this framework.

There is also prior work on using neural network models in BO for hyperparameter optimization (Snoek et al. 2015; Springenberg et al. 2016), The explicit goal of these papers is to improve the efficiency of Gaussian process-based BO from cubic to linear time, not to develop a different type of prediction model in order to improve the performance of BO with respect to the number of iterations. We provide additional related work details in the full version of the paper.

Subsequent work. Since its release, several papers have included BANANAS in new experiments, further showing that BANANAS is a competitive NAS algorithm (Krishna et al. 2020; Siems et al. 2020; Nguyen et al. 2020; Ru et al. 2020; Wei et al. 2020). Finally, a recent paper conducted a study on several encodings used for NAS (White et al. 2020), concluding that neural predictors perform well with the path encoding.

BO + Neural Predictor Framework

In this section, we give a background on BO, and we describe the BO + neural predictor framework. In applications of BO for deep learning, the typical goal is to find a neural architecture and/or set of hyperparameters that lead to an optimal validation error. Formally, BO seeks to compute $a^* = \arg\min_{a \in A} f(a)$, where A is the search space, and $f(a)$ denotes the validation error of architecture a after training on a fixed dataset for a fixed number of epochs. In the standard BO setting, over a sequence of iterations, the results from all previous iterations are used to model the topology of $\{f(a)\}_{a \in A}$ using the posterior distribution of the model (often a GP). The next architecture is then chosen by optimizing an acquisition function such as expected improvement (EI) (Močkus 1975) or Thompson sampling (TS) (Thompson 1933). These functions balance exploration with exploitation during the search. The chosen architecture is then trained and

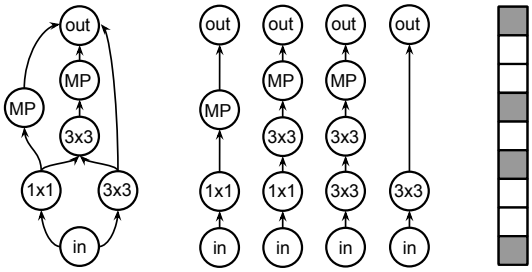

Figure 1: A neural architecture (left), is decomposed into a set of its paths from input to output (middle), which is then encoded as a one-hot vector (right).

used to update the model of $\{f(a)\}_{a \in A}$. Evaluating $f(a)$ in each iteration is the bottleneck of BO (since a neural network must be trained). To mitigate this, parallel BO methods typically output k architectures to train in each iteration, so that the k architectures can be trained in parallel.

BO + neural predictor framework. In each iteration of BO, we train a neural network on all previously evaluated architectures, a, to predict the validation accuracy $f(a)$ of unseen architectures. The architectures are represented as labeled DAGs (Ying et al. 2019; Dong and Yang 2020), and there are different methods of encoding the DAGs before they are passed to the neural predictor (Ying et al. 2019; White et al. 2020), which we describe in the next section. Choices for the neural predictor include feedforward networks, graph convolutional networks (GCN), and variational autoencoder (VAE)-based networks. In order to evaluate an acquisition function, we also compute an uncertainty estimate for each input datapoint. This can be accomplished by using, for example, a Bayesian neural network or an ensemble of neural predictors. Given the acquisition function, an optimization routine is then carried out, which returns the next architecture to be evaluated. In the next section, we give a thorough analysis of the choices that must be made when instantiating this framework.

Analysis of the Framework

In this section, we give an extensive study of the BO + neural predictor framework. First, we discuss architecture encodings, and we define a novel featurization called the path encoding. Then we conduct an analysis of different choices of neural predictors. Next, we analyze different methods for achieving calibrated uncertainty estimates from the neural predictors. After that, we conduct experiments on different acquisition functions and acquisition function optimization routines. Finally, we use these analyses to create our algorithm, BANANAS.

Throughout this section, we run experiments on the NASBench-101 dataset (experiments on additional search spaces are given in the full-length paper). The NASBench-101 dataset (Ying et al. 2019) consists of over 423,000 neural architectures from a cell-based search space, and each architecture comes with precomputed validation and test accuracies on CIFAR-10. The search space consists of a DAG with 7 nodes that can each take on three different operations, and there can be at most 9 edges between the nodes. We use the open source version of the NASBench-101 dataset (Ying et al. 2019). We give the full details about the use of NASBench-101 in the full version of the paper. Our code is available at https://github.com/naszilla/naszilla.

Architecture encodings. The majority of existing work on neural predictors use an adjacency matrix representation to encode the neural architectures. The adjacency matrix encoding gives an arbitrary ordering to the nodes, and then gives a binary feature for an edge between node i and node j, for all $i < j$. Then a list of the operations at each node must also be included in the encoding. This is a challenging data structure for a neural predictor to interpret because it relies on an arbitrary indexing of the nodes, and features are highly dependent on one another. For example, an edge from the input to node 2 is useless if there is no path from node 2 to the output. And if there *is* an edge from node 2 to the output, this edge is highly correlated with the feature that describes the operation at node 2 (conv_1x1, pool_3x3, etc.). A continuous-valued variant of the adjacency matrix encoding has also been tested (Ying et al. 2019).

We introduce a novel encoding which we term the *path encoding*, and we show that it substantially increases the performance of neural predictors. The path encoding is quite simple to define: there is a binary feature for each path from the input to the output of an architecture cell, given in terms of the operations (e.g., input→conv_1x1→pool_3x3→output). To encode an architecture, we simply check which paths are present in the architecture, and set the corresponding features to 1s. See Figure 1. Intuitively, the path encoding has a few strong advantages. The features are not nearly as dependent on one another as they are in the adjacency matrix encoding, since each feature represents a unique path that the data tensor can take from the input node to the output node. Furthermore, there is no longer an arbitrary node ordering, which means that each neural architecture maps to only one encoding (which is not true for the adjacency matrix encoding). On the other hand, it is possible for multiple architectures to map to the same path encoding (i.e., the encoding is well-defined, but it is not one-to-one). However, subsequent work showed that architectures with the same path encoding also have very similar validation errors (White et al. 2020), which is beneficial in NAS algorithms.

The length of the path encoding is the total number of possible paths in a cell, $\sum_{i=0}^{n} q^i$, where n denotes the number of nodes in the cell, and q denotes the number of operations for each node. However, we present theoretical and experimental evidence that substantially truncating the path encoding, even to length smaller than the adjacency matrix encoding, does not decrease its performance. Many NAS algorithms sample architectures by randomly sampling edges in the DAG subject to a maximum edge constraint (Ying et al. 2019). Intuitively, the vast majority of paths have a very low probability of occurring in a cell returned by this procedure. Therefore,

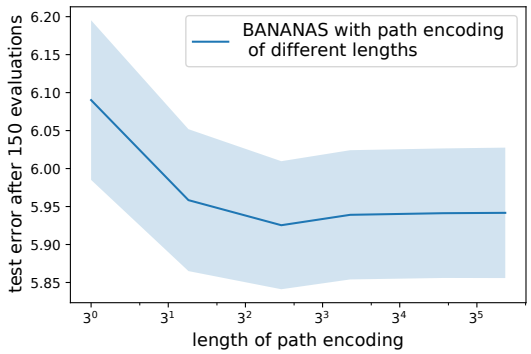

Figure 2: Performance of BANANAS with the path encoding truncated to different lengths. Since each node has 3 choices of operations, the "natural" cutoffs are at powers of 3.

by simply truncating the least-likely paths, our encoding scales *linearly* in the size of the cell, with an arbitrarily small amount of information loss. In the following theorem, let $G_{n,k,r}$ denote a DAG architecture with n nodes, r choices of operations on each node, and where each potential forward edge ($\frac{n(n-1)}{2}$ total) was chosen with probability $\frac{2k}{n(n-1)}$ (so that the expected number of edges is k).

Theorem 0.1 (informal). *Given integers $r, c > 0$, there exists an N such that $\forall\ n > N$, there exists a set of n paths $\mathcal{P}'$ such that the probability that $G_{n,n+c,r}$ contains a path not in $\mathcal{P}'$ is less than $\frac{1}{n^2}$.*

For the formal statement and full proof, see the full version of the paper. This theorem says that when n is large enough, with high probability, we can truncate the path encoding to a size of just n without losing information. Although the asymptotic nature of this result makes it a proof of concept, we empirically show in Figure 2 that in BANANAS running on NASBench-101, the path encoding can be truncated from its full size of $\sum_{i=0}^{5} 3^i = 364$ bits to a length of just *twenty* bits, without a loss in performance. (The exact experimental setup for this result is described later in this section.) In fact, the performance after truncation actually *improves* up to a certain point. We believe this is because with the full-length encoding, the neural predictor overfits to very rare paths. In the full version of the paper, we show a similar result for NASBench-201 (Dong and Yang 2020): the full path encoding length of $\sum_{i=0}^{3} 5^i = 156$ can be truncated to just 30, without a loss of performance.

Neural predictors. Now we study the neural predictor, a crucial component in the BO + neural predictor framework. Recall from the previous section that a neural predictor is a neural network that is repeatedly trained on the current set of evaluated neural architectures and predicts the accuracy of unseen neural architectures. Prior work has used GCNs (Shi et al. 2019; Ma, Cui, and Yang 2019) or VAE-based architectures (Zhang et al. 2019) for this task. We evaluate the performance of standard feedfoward neural networks with either the adjacency matrix or path-based encoding, compared to VAEs and GCNs in predicting the validation accuracy of neural architectures. The feedforward neural network we use is a sequential fully-connected network with 10 layers of width 20, the Adam optimizer with a learning rate of 0.01, and the loss function set to mean absolute error (MAE). We use open-source implementations of the GCN [2] and VAE (Zhang et al. 2019). See the full version of the paper for a full description of our implementations.

In Figure 3 (left), we compare the different neural predictors by training them on a set of neural architectures drawn i.i.d. from NASBench-101, along with validation accuracies, and then computing the MAE on a held-out test set of size 1000. We run 50 trials for different training set sizes and average the results. The best performing neural predictors are the feedforward network with the path encoding (with and without truncation) and the GCN. The feedforward networks also had shorter runtime compared to the GCN and VAE, however, the runtime of the full NAS algorithm is dominated by evaluating neural architectures, not by training neural predictors.

Uncertainty calibration. In the previous section, we evaluated standalone neural predictors. To incorporate them within BO, for any datapoint, neural predictors need to output both a prediction and an uncertainty estimate for that prediction. Two popular ways of achieving uncertainties are by using a Bayesian neural network (BNN), or by using an ensemble of neural predictors. In a BNN, we infer a posterior distribution over network weights. It has been demonstrated recently that accurate prediction and uncertainty estimates in neural networks can be achieved using Hamiltonian Monte Carlo (Springenberg et al. 2016). In the ensemble approach, we train m neural predictors using different random weight initializations and training set orders. Then for any datapoint, we can can compute the mean and standard deviation of these m predictions. Ensembles of neural networks, even of size three and five, have been shown in some cases to give more reliable uncertainty estimates than other leading approaches such as BNNs (Lakshminarayanan, Pritzel, and Blundell 2017; Beluch et al. 2018; Choi et al. 2016; Snoek et al. 2019; Zaidi et al. 2020).

We compare the uncertainty estimate of a BNN with an ensemble of size five for each of the neural predictors described in the previous section. We use the BOHAMIANN implementation for the BNN (Springenberg et al. 2016), and to ensure a fair comparison with the ensembles, we train it for five times longer. The experimental setup is similar to the previous section, but we compute a standard measure of calibration: root mean squared calibration error (RMSCE) on the test set (Kuleshov, Fenner, and Ermon 2018; Tran et al. 2020). See Figure 3 (middle). Intuitively, the RMSCE is low if a method yields a well-calibrated predictive estimate (i.e. predicted coverage of intervals equals the observed coverage). All ensemble-based predictors yielded better uncertainty estimates than the BNN, consistent with prior work. Note that RMSCE only measures the quality of uncertainty estimates, agnostic to prediction accuracy. We must therefore look at

[2] https://github.com/ultmaster/neuralpredictor.pytorch

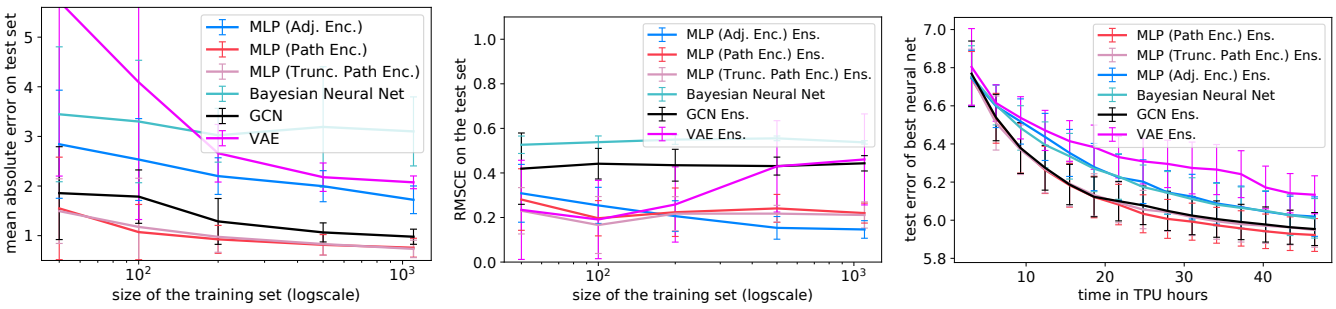

Figure 3: Performance of neural predictors on NASBench-101: predictive ability (left), accuracy of uncertainty estimates (middle), performance in NAS when combined with BO (right).

prediction (Figure 3 left) and RMSCE (Figure 3 middle) together when evaluating the neural predictors.

Finally, we evaluate the performance of each neural predictor within the full BO + neural predictor framework. We use the approach described in the previous section, using independent Thompson sampling and mutation for acquisition optimization (described in more detail in the next section). Each algorithm is given a budget of 47 TPU hours, or about 150 neural architecture evaluations on NASBench-101. That is, there are 150 iterations of training a neural predictor and choosing a new architecture to evaluate using the acquisition function. The algorithms output 10 architectures in each iteration of BO for better parallelization, as described in the previous section. After each iteration, we return the test error of the architecture with the best validation error found so far. We run 200 trials of each algorithm and average the results. This is the same experimental setup as in Figure 2, as well as experiments later in this section and the next section. See Figure 3 (right). The two best-performing neural predictors are an ensemble of GCNs, and an ensemble of feedforward neural networks with the path encoding, with the latter having a slight edge. The feedforward network is also desirable because it requires less hyperparameter tuning than the GCN.

Acquisition functions and optimization. Now we analyze the BO side of the framework, namely, the choice of acquisition function and optimization strategy. We consider four common acquisition functions that can be computed using a function value estimate and uncertainty estimate for each input datapoint: Thompson sampling (TS) (Thompson 1933), upper confidence bound (UCB) (Srinivas et al. 2009), expected improvement (EI) (Močkus 1975), and probability of improvement (PI) (Kushner 1964). We also consider a variant of TS called independent Thompson sampling (ITS), which uses a unique posterior function sample for each input architecture. Later, we show that ITS has strong empirical performance.

First we give the formal definitions of each acquisition function. Suppose we have trained an ensemble of M predictive models, $\{f_m\}_{m=1}^M$, where $f_m : A \to \mathbb{R}$ for all m. Let $y_{\min}$ denote the lowest validation error of an architecture discovered so far. Following previous work (Neiswanger et al. 2019), we use the following acquisition function estimates for an input architecture $a \in A$:

$$\phi_{\text{TS}}(x) = f_{\tilde{m}}(x), \quad \tilde{m} \sim \text{Unif}\,(1, M) \tag{1}$$

$$\phi_{\text{ITS}}(x) = \tilde{f}_x(x), \quad \tilde{f}_x(x) \sim \mathcal{N}(\hat{f}, \hat{\sigma}^2) \tag{2}$$

$$\phi_{\text{UCB}}(x) = \hat{f} - \beta\hat{\sigma} \tag{3}$$

$$\begin{aligned}\phi_{\text{EI}}(a) &= \mathbb{E}\left[\mathbb{1}\left[f_m(a) > y_{\min}\right](y_{\min} - f_m(a))\right] \\ &= \int_{-\infty}^{y_{\min}} (y_{\min} - y)\,\mathcal{N}\left(\hat{f}, \hat{\sigma}^2\right) dy\end{aligned} \tag{4}$$

$$\begin{aligned}\phi_{\text{PI}}(x) &= \mathbb{E}\left[\mathbb{1}\left[f_m(x) > y_{\min}\right]\right] \\ &= \int_{-\infty}^{y_{\min}} \mathcal{N}\left(\hat{f}, \hat{\sigma}^2\right) dy\end{aligned} \tag{5}$$

In these acquisition function definitions, $\mathbb{1}(x) = 1$ if x is true and 0 otherwise, and we are making a normal approximation for our model's posterior predictive density, where we estimate parameters

$$\hat{f} = \frac{1}{M}\sum_{m=1}^{M} f_m(x), \text{ and } \hat{\sigma} = \sqrt{\frac{\sum_{m=1}^{M}(f_m(x) - \hat{f})^2}{M-1}}.$$

In the UCB acquisition function experiments, we set the tradeoff parameter $\beta = 0.5$. We tested each acquisition function within the BO + neural predictor framework, using mutation for acquisition function optimization and the best neural predictor from the previous section - an ensemble of feedforward networks with the path encoding. The experimental setup is the same as in previous sections. See Figure 5 (left). We see that the acquisition function does not have as big an effect on performance as other components, though ITS performs the best overall. Note also that both TS and ITS have advantages when running parallel experiments, since they are stochastic acquisition functions that can be directly applied in the batch BO setting (Kandasamy et al. 2018a).

Next, we test different acquisition function optimization strategies. In each iteration of BO, our goal is to find the neural architecture from the search space which minimizes the acquisition function. Evaluating the acquisition function for every neural architecture in the search space is computationally infeasible. Instead, we create a set of 100-1000 architectures (potentially in an iterative fashion) and choose the architecture with the value of the acquisition function in this set.

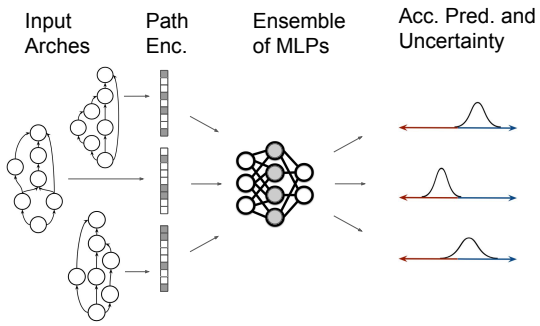

Figure 4: Diagram of the BANANAS neural predictor.

The simplest strategy is to draw 1000 random architectures. However, it can be beneficial to generate a set of architecture that are close in edit distance to architectures in the training set, since the neural predictor is more likely to give accurate predictions to these architectures. Furthermore, local optimization methods such as mutation, evolution, and local search have been shown to be effective for acquisition function optimization (Balandat et al. 2019; Kandasamy et al. 2018b; Wilson, Hutter, and Deisenroth 2018). In "mutation", we simply mutate the architectures with the best validation accuracy that we have found so far by randomly changing one operation or one edge. In local search, we iteratively take the architectures with the current highest acquisition function value, and compute the acquisition function of all architectures in their neighborhood. In evolution, we iteratively maintain a population by mutating the architectures with the highest acquisition function value and killing the architectures with the lowest values. We give the full details of these methods in the full version of the paper. The experimental setup is the same as in the previous sections. See Figure 5 (middle). We see that mutation performs the best, which indicates that it is better to consider architectures with edit distance closer to the set of already evaluated architectures.

BANANAS: Bayesian optimization with neural architectures for NAS. Using the best components from the previous sections, we construct our full NAS algorithm, BANANAS, composed of an ensemble of feedforward neural networks using the path encoding, ITS, and a mutation acquisition function. See Algorithm 1 and Figure 4. Note that in the previous sections, we conducted experiments on each component individually while keeping all other components fixed. In the full version of the paper, we give further analysis varying all components at once, to ensure that BANANAS is indeed the optimal instantiation of this framework.

For the loss function in the neural predictors, we use mean absolute percentage error (MAPE) because it gives a higher weight to architectures with lower validation losses:

$$\mathcal{L}(y_{\text{pred}}, y_{\text{true}}) = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_{\text{pred}}^{(i)} - y_{\text{LB}}}{y_{\text{true}}^{(i)} - y_{\text{LB}}} - 1\right|, \qquad (6)$$

where $y_{\text{pred}}^{(i)}$ and $y_{\text{true}}^{(i)}$ are the predicted and true values of the validation error for architecture i, and y_{LB} is a global lower bound on the minimum true validation error. To parallelize Algorithm 1, in step iv. we simply choose the k architectures with the smallest values of the acquisition function and evaluate the architectures in parallel.

Algorithm 1 BANANAS

Input: Search space A, dataset D, parameters t_0, T, M, c, x, acquisition function ϕ, function $f(a)$ returning validation error of a after training.

1. Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from A and train them on D.
2. For t from t_0 to T,
   i. Train an ensemble of neural predictors on $\{(a_0, f(a_0)), \ldots, (a_t, f(a_t))\}$ using the path encoding to represent each architecture.
   ii. Generate a set of c candidate architectures from A by randomly mutating the x architectures a from $\{a_0, \ldots, a_t\}$ that have the lowest value of $f(a)$.
   iii. For each candidate architecture a, evaluate the acquisition function $\phi(a)$.
   iv. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$, and evaluate $f(a_{t+1})$.

Output: $a^* = \text{argmin}_{t=0,\ldots,T} f(a_t)$.

BANANAS Experiments

In this section, we compare BANANAS to many other popular NAS algorithms on three search spaces. To promote reproducibility, we discuss our adherence to the NAS research checklist (Lindauer and Hutter 2019) in the full version of the paper. In particular, we release our code, we use a tabular NAS dataset, and we run many trials of each algorithm.

We run experiments on NASBench-101 described in the previous section, as well as NASBench-201 and the DARTS search space. The NASBench-201 dataset (Yang, Esperança, and Carlucci 2020) consists of 15625 neural architectures with precomputed validation and test accuracies for 200 epochs on CIFAR-10, CIFAR-100, and ImageNet-16-120. The search space consists of a complete directed acyclic graph on 4 nodes, and each edge can take on five different operations. The DARTS search space (Liu, Simonyan, and Yang 2018) is size 10^{18}. It consists of two cells: a convolutional cell and a reduction cell. Each cell has four nodes that have two incoming edges which take on one of eight operations.

Performance on NASBench search spaces. We compare BANANAS to various popular NAS algorithms: random search (Li and Talwalkar 2019), regularized evolution (Real et al. 2019), BOHB (Falkner, Klein, and Hutter 2018), NASBOT (Kandasamy et al. 2018b), local search (White, Nolen, and Savani 2020), TPE (Bergstra et al. 2011), BOHAMIANN (Springenberg et al. 2016), BONAS (Shi et al. 2019), REINFORCE (Williams 1992), GP-based BO (Snoek, Larochelle, and Adams 2012), AlphaX (Wang et al. 2018), GCN Predictor (Wen et al. 2019), and DNGO (Snoek et al. 2015). As much as possible, we use the code directly from the

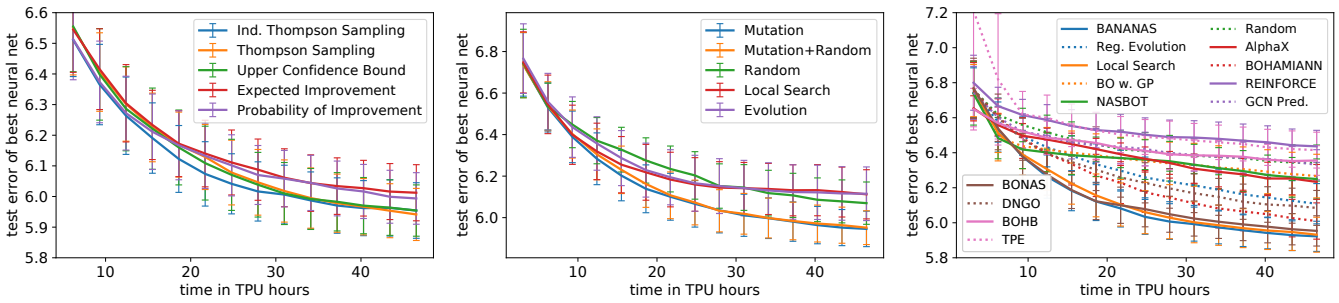

Figure 5: Performance of different acquisition functions (left). Performance of different acquisition function optimization strategies (middle). Performance of BANANAS compared to other NAS algorithms (right). See the full version of the paper for the same results in a table.

NAS Algorithm	Source	Avg. Test error	Runtime	Method
Random search	(Liu, Simonyan, and Yang 2018)	3.29	4	Random
Local search	(White, Nolen, and Savani 2020)	3.49	11.8	Local search
DARTS	(Liu, Simonyan, and Yang 2018)	2.76	5	Gradient-based
ASHA	(Li and Talwalkar 2019)	3.03	9	Successive halving
Random search WS	(Li and Talwalkar 2019)	2.85	9.7	Random
DARTS	Ours	2.68	5	Gradient-based
ASHA	Ours	3.08	9	Successive halving
BANANAS	Ours	**2.64**	11.8	BO + neural predictor

Table 1: Comparison of NAS algorithms on the DARTS search space. The runtime unit is total GPU-days on a Tesla V100.

open-source repositories, without changing the hyperparameters (but with a few exceptions). For a description of each algorithm and details of the implementations we used, see the full version of the paper. The experimental setup is the same as in the previous section. For results on NASBench-101, see Figure 5 (right). The top three algorithms in order, are BANANAS, local search, and BONAS. In the full version of the paper, we also show that BANANAS achieves strong performance on the three datasets in NASBench-201.

Performance on the DARTS search space. We test BANANAS on the search space from DARTS. Since the DARTS search space is not a tabular dataset, we cannot fairly compare to other methods which use substantially different training and testing pipelines (Lindauer and Hutter 2019). We use a common evaluation pipeline which is to train for 600 epochs with cutout and auxiliary tower (Liu, Simonyan, and Yang 2018; Li and Talwalkar 2019; Yan et al. 2020), where the state of the art is around 2.6% on CIFAR-10. Other papers use different evaluation settings (e.g., training for many more epochs) to achieve lower error, but they cannot be fairly compared to other algorithms.

In our experiments, BANANAS is given a budget of 100 evaluations. In each evaluation, the chosen architecture is trained for 50 epochs and the average validation error of the last 5 epochs is recorded. To ensure a fair comparison by controlling all hyperparameter settings and hardware, we re-trained the architectures from prior work when they were available. In this case, we report the mean test error over five random seeds of the best architecture found for each method. We compare BANANAS to DARTS (Liu, Simonyan, and Yang 2018), random search (Liu, Simonyan, and Yang 2018), local search (White, Nolen, and Savani 2020), and ASHA (Li and Talwalkar 2019). See Table 1. A new surrogate benchmark on the DARTS search space (Siems et al. 2020), called NASBench-301 was recently introduced. Initial experiments showed (Siems et al. 2020) that BANANAS was competitive with nine other popular NAS algorithms, including one-shot methods (Liu, Simonyan, and Yang 2018; Xu et al. 2019; Dong and Yang 2019).

Conclusion and Future Work

We conduct an analysis of the BO + neural predictor framework, which has recently emerged as a high-performance framework for NAS. We test several methods for each component: the encoding, neural predictor, calibration method, acquisition function, and acquisition function optimization strategy. We also propose a novel path-based encoding scheme, which improves the performance of neural predictors. We use all of this analysis to develop BANANAS, an instantiation of the BO + neural predictor framework which achieves state-of-the-art performance on popular NAS search spaces. Interesting follow-up ideas are to develop multi-fidelity or successive halving versions of BANANAS. Incorporating these approaches with BANANAS could result in a significant decrease in the runtime without sacrificing accuracy.

Acknowledgments

We thank Jeff Schneider, Naveen Sundar Govindarajulu, and Liam Li for their help with this project. WN was supported by U.S. Department of Energy Office of Science under Contract No. DE-AC02-76SF00515.

Ethical Statement

Our work gives a new method for neural architecture search, with the aim of improving the performance of future deep learning research. Therefore, we have much less control over the net impact of our work on society. For example, our work may be used to tune a deep learning optimizer for reducing the carbon footprint of large power plants, but it could just as easily be used to improve a deep fake generator. Clearly, the first example would have a positive impact on society, while the second example may have a negative impact.

Our work is one level of abstraction from real applications, but our algorithm, and more generally the field of NAS, may become an important step in advancing the field of artificial intelligence. Because of the recent push for explicitly reasoning about the impact of research in AI (Hecht et al. 2018), we are hopeful that neural architecture search will be used to benefit society.

References

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

Balandat, M.; Karrer, B.; Jiang, D. R.; Daulton, S.; Letham, B.; Wilson, A. G.; and Bakshy, E. 2019. BoTorch: Programmable Bayesian Optimization in PyTorch. *arXiv preprint arXiv:1910.06403* .

Beluch, W. H.; Genewein, T.; Nürnberger, A.; and Köhler, J. M. 2018. The power of ensembles for active learning in image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Choi, Y.; Kwon, Y.; Lee, H.; Kim, B. J.; Paik, M. C.; and Won, J.-H. 2016. Ensemble of deep convolutional neural networks for prognosis of ischemic stroke. In *International Workshop on Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*.

Deng, B.; Yan, J.; and Lin, D. 2017. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351* .

Dong, X.; and Yang, Y. 2019. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, 1761–1770.

Dong, X.; and Yang, Y. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* .

Falkner, S.; Klein, A.; and Hutter, F. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Golovin, D.; Solnik, B.; Moitra, S.; Kochanski, G.; Karro, J.; and Sculley, D. 2017. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Hecht, B.; Wilcox, L.; Bigham, J. P.; Schöning, J.; Hoque, E.; Ernst, J.; Bisk, Y.; De Russis, L.; Yarosh, L.; Anjum, B.; Contractor, D.; and Wu, C. 2018. It's time to do something: Mitigating the negative impacts of computing through a change to the peer review process. *ACM Future of Computing Blog* .

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Istrate, R.; Scheidegger, F.; Mariani, G.; Nikolopoulos, D.; Bekas, C.; and Malossi, A. C. I. 2019. Tapas: Train-less accuracy predictor for architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Jin, H.; Song, Q.; and Hu, X. 2018. Auto-keras: Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282* .

Kandasamy, K.; Krishnamurthy, A.; Schneider, J.; and Póczos, B. 2018a. Parallelised bayesian optimisation via thompson sampling. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

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

Kitano, H. 1990. Designing neural networks using genetic algorithms with graph generation system. *Complex systems* 4(4): 461–476.

Klein, A.; Falkner, S.; Springenberg, J. T.; and Hutter, F. 2017. Learning curve prediction with Bayesian neural networks. *ICLR 2017* .

Krishna, C. S.; Gupta, A.; Rai, H.; and Narayan, S. 2020. Neural Architecture Search with Reinforce and Masked Attention Autoregressive Density Estimators. *arXiv preprint arXiv:2006.00939* .

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Kuleshov, V.; Fenner, N.; and Ermon, S. 2018. Accurate uncertainties for deep learning using calibrated regression. *arXiv preprint arXiv:1807.00263* .

Kushner, H. J. 1964. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering* 86(1): 97–106.

Lakshminarayanan, B.; Pritzel, A.; and Blundell, C. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, 6402–6413.

Li, L.; and Talwalkar, A. 2019. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638* .

Lindauer, M.; and Hutter, F. 2019. Best Practices for Scientific Research on Neural Architecture Search. *arXiv preprint arXiv:1909.02453* .

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

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* .

Ma, L.; Cui, J.; and Yang, B. 2019. Deep neural architecture search with deep graph bayesian optimization. In *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, 500–507. IEEE.

Maziarz, K.; Khorlin, A.; de Laroussilhe, Q.; and Gesmundo, A. 2018. Evolutionary-Neural Hybrid Agents for Architecture Search. *arXiv preprint arXiv:1811.09828* .

Močkus, J. 1975. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, 400–404. Springer.

Neiswanger, W.; Kandasamy, K.; Poczos, B.; Schneider, J.; and Xing, E. 2019. Probo: a framework for using probabilistic programming in bayesian optimization. *arXiv preprint arXiv:1901.11515* .

Nguyen, V.; Le, T.; Yamada, M.; and Osborne, M. A. 2020. Optimal transport kernels for sequential and parallel neural architecture search. *arXiv preprint arXiv:2006.07593* .

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* .

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*.

Ru, B.; Wan, X.; Dong, X.; and Osborne, M. 2020. Neural Architecture Search using Bayesian Optimisation with Weisfeiler-Lehman Kernel. *arXiv preprint arXiv:2006.07556* .

Shi, H.; Pi, R.; Xu, H.; Li, Z.; Kwok, J. T.; and Zhang, T. 2019. Multi-objective Neural Architecture Search via Predictive Network Performance Optimization. *arXiv preprint arXiv:1911.09336* .

Siems, J.; Zimmer, L.; Zela, A.; Lukasik, J.; Keuper, M.; and Hutter, F. 2020. NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search. *arXiv preprint arXiv:2008.09777* .

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Snoek, J.; Ovadia, Y.; Fertig, E.; Lakshminarayanan, B.; Nowozin, S.; Sculley, D.; Dillon, J.; Ren, J.; and Nado, Z. 2019. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

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

Springenberg, J. T.; Klein, A.; Falkner, S.; and Hutter, F. 2016. Bayesian optimization with robust Bayesian neural networks. In *Advances in Neural Information Processing Systems*, 4134–4142.

Srinivas, N.; Krause, A.; Kakade, S. M.; and Seeger, M. 2009. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995* .

Stanley, K. O.; and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10(2): 99–127.

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. A. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25(3/4): 285–294.

Tran, K.; Neiswanger, W.; Yoon, J.; Zhang, Q.; Xing, E.; and Ulissi, Z. W. 2020. Methods for comparing uncertainty quantifications for material property predictions. *Machine Learning: Science and Technology* 1(2): 025006.

Wang, L.; Zhao, Y.; Jinnai, Y.; and Fonseca, R. 2018. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1805.07440* .

Wei, C.; Niu, C.; Tang, Y.; and Liang, J. 2020. NPENAS: Neural Predictor Guided Evolution for Neural Architecture Search. *arXiv preprint arXiv:2003.12857* .

Wen, W.; Liu, H.; Li, H.; Chen, Y.; Bender, G.; and Kindermans, P.-J. 2019. Neural Predictor for Neural Architecture Search. *arXiv preprint arXiv:1912.00848* .

White, C.; Neiswanger, W.; Nolen, S.; and Savani, Y. 2020. A study on encodings for neural architecture search. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

White, C.; Nolen, S.; and Savani, Y. 2020. Local Search is State of the Art for NAS Benchmarks. *arXiv preprint arXiv:2005.02960* .

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 229–256.

Wilson, J.; Hutter, F.; and Deisenroth, M. 2018. Maximizing acquisition functions for Bayesian optimization. In *Advances in Neural Information Processing Systems*, 9884–9895.

Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.-J.; Tian, Q.; and Xiong, H. 2019. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*.

Yan, S.; Zheng, Y.; Ao, W.; Zeng, X.; and Zhang, M. 2020. Does unsupervised architecture representation learning help neural architecture search? *arXiv preprint arXiv:2006.06936* .

Yang, A.; Esperança, P. M.; and Carlucci, F. M. 2020. NAS evaluation is frustratingly hard. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ying, C.; Klein, A.; Real, E.; Christiansen, E.; Murphy, K.; and Hutter, F. 2019. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635* .

Zaidi, S.; Zela, A.; Elsken, T.; Holmes, C.; Hutter, F.; and Teh, Y. W. 2020. Neural ensemble search for performant and calibrated predictions. *arXiv preprint arXiv:2006.08573* .

Zhang, C.; Ren, M.; and Urtasun, R. 2018. Graph hypernetworks for neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Zhang, M.; Jiang, S.; Cui, Z.; Garnett, R.; and Chen, Y. 2019. D-VAE: A variational autoencoder for directed acyclic graphs. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; and Sun, M. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* .

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.