Neural Architecture Search via Bayesian Optimization with a Neural Network Model

Colin White
RealityEngines.AI
colin@realityengines.ai

Willie Neiswanger
Carnegie Mellon University
willie@cs.cmu.edu

Yash Savani
RealityEngines.AI
yash@realityengines.ai

Abstract

Neural Architecture Search (NAS) has seen an explosion of research in the past three years. A variety of methods have been proposed to perform NAS, including reinforcement learning, Bayesian optimization with a Gaussian process prior, evolutionary search, and gradient descent. In this work, we design a NAS algorithm which uses Bayesian optimization with a neural network prediction model.

We use a path-based encoding scheme to featurize the neural architectures that are used as training data for the meta neural network. This method is particularly effective for encoding architectures in cell-based search spaces. After training on just 150 random neural architectures, we are able to predict the validation accuracy of an architecture to within one percent of its true accuracy on average. This may be of independent interest beyond Bayesian neural architecture search.

We test our algorithm on the NAS-Bench-101 dataset [Ying et al. 2019], and we show that our algorithm significantly outperforms baselines including evolutionary search and reinforcement learning. We try several acquisition functions and show that the upper confidence bound function works the best. Finally, we show our path-based encoding scheme significantly improves the performance of the NAS algorithm compared to other encoding methods. [1]

1 Introduction

Since the deep learning revolution in 2012, neural networks have been getting more specialized and more complex [Krizhevsky et al., 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. However, a new area of research, neural architecture search (NAS), seeks to automate this process. Since the popular NASNet paper [Zoph and Le, 2017], there has been a flurry of research on neural architecture search [Liu et al., 2018a, Pham et al., 2018, Liu et al., 2018b, Kandasamy et al., 2018, Elsken et al., 2018, Jin et al., 2018, Sciuto et al., 2019, Cortes et al., 2017, Baker et al., 2016, Liu et al., 2017]. Many methods have been proposed for NAS, including random search, evolutionary search, reinforcement learning, Bayesian optimization, and gradient descent. Each technique has its advantages and disadvantages. For example, reinforcement learning with a controller neural network can learn very complex policies. However, as some past works have argued, neural architecture search is an optimization problem - find the architecture with the lowest validation error - so maintaining a state and transition function may be overkill [Kandasamy et al., 2018, Elsken et al., 2018]. On the other hand, Bayesian optimization with a Gaussian process prior is a leading method for zero-th order optimization of functions which are expensive to evaluate. This is exactly the setting of deep learning model selection. Although Bayesian optimization has seen great success in hyperparameter optimization of deep learning functions [Golovin et al., 2017, Li et al., 2016], a host of problems arise

[1] This work was extended to a full-length paper here: `https://arxiv.org/abs/1910.11858`. All of the code for this work is available here: `https://github.com/naszilla/bananas`.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

when using Bayesian optimization for neural architecture search. For example, while there is a huge body of work on Bayesian optimization over Euclidean space, using Bayesian optimization for NAS requires choosing a distance function between neural architectures. This is often a cumbersome task involving hyper-hyperparameter tuning for parameters of the distance function [Kandasamy et al., 2018, Jin et al., 2018]. Furthermore, Bayesian optimization is most commonly set up with a Gaussian process prior, and it can be quite challenging to find a kernel that is expressive enough to predict the performance of neural networks [Elsken et al., 2018].

In this work, we run Bayesian optimization with a neural network prediction model. That is, in every iteration of Bayesian optimization, we train a meta neural network to predict the accuracy of each neural network in our search space. We use this meta neural network to guide the search procedure. This avoids the aforementioned problems with Bayesian optimization NAS: the model is powerful enough to predict neural network accuracies, and there is no need to construct a distance function between neural networks by hand.

Training a meta neural network to predict the accuracy of neural networks is a challenging task. The majority of popular NAS algorithms are deployed over a directed acyclic gaph (DAG) search space – the set of all possible architectures to choose from is the set of all DAGs of a certain size, together with all possible combinations of operations on each node [Zoph and Le, 2017, Pham et al., 2018, Liu et al., 2018b, Ying et al., 2019]. This poses a roadblock when using a meta neural network to predict the accuracies of neural networks, since graph structures are difficult for neural networks to learn [Zhou et al., 2018]. We use a *path-based encoding scheme* to encode a neural network, which drastically improves the accuracy of our meta neural network. For each neural network, we enumerate all possible paths along the DAG from the input layer to the output layer, and we create a binary feature for all possible paths. This method improves the accuracy of our meta neural network by a factor of 3. With just 150 training points, our meta neural network is able to predict the accuracy of unseen neural networks to within one percent on average on the nasbench dataset [Ying et al., 2019]. This may be of interest beyond Bayesian neural architecture search.

We develop a NAS algorithm by running Bayesian optimization with a meta neural network model. We use an ensemble method to predict the mean and variance of candidate neural networks, from which we compute upper confidence bounds (UCB) as our acquisition function. Finally, we use a mutation function to optimize the acquisition function. We compare our NAS algorithm against a host of benchmark algorithms on the nasbench dataset including random search, regularized evolution, and reinforcement learning. Our method sees significant improvements compared to the benchmark algorithms. We test acquisition functions including Thompson sampling, expected improvement, probability of improvement, and UCB, and we show that UCB performs the best. We are committed to fair, reproducible NAS research, and we address all of the points in the recent NAS research checklist [Lindauer and Hutter, 2019].

2 Related Work

Neural architecture search Neural architecture search has been studied since at least the 1990s [Floreano et al., 2008, Kitano, 1990, Stanley and Miikkulainen, 2002]. NASNet, a reinforcement learning algorithm, was the first neural architecture search algorithm to gain significant attention in recent years [Zoph and Le, 2017]. Some of the most popular recent techniques for NAS include evolutionary algorithms [Shah et al., 2018], reinforcement learning [Zoph and Le, 2017, Pham et al., 2018, Liu et al., 2018a, Tan and Le, 2019], Bayesian optimization [Kandasamy et al., 2018, Jin et al., 2018], and gradient descent [Liu et al., 2018b]. Recent papers have highlighted the need for fair and reproducible NAS comparisons [Li and Talwalkar, 2019, Sciuto et al., 2019, Lindauer and Hutter, 2019], and then nasbench dataset was created for this purpose [Ying et al., 2019]. There are several works which predict the validation accuracy of neural networks [Deng et al., 2017, Istrate et al., 2019], or the curve of validation accuracy with respect to training time [Klein et al., 2017, Domhan et al., 2015, Baker et al., 2017]. A recent algorithm, AlphaX, uses a meta neural network to perform NAS [Wang et al., 2018], which is a similar high-level idea to our work but with many key differences. Their search is progressive, and each iteration makes a small change to the current neural network, rather than choosing a completely new neural network. Therefore, their meta neural network is trained to optimize the selection of new actions as well as predict the performance of the new progressive network. Furthermore, they use an adjacency matrix featurization for the inputs to

the meta neural network, which we find to perform worse than our path-based encoding. For a more detailed discussion on related work, see Appendix A.

3 Methodology

Search space. In this work, we consider convolutional cell-based search spaces [Zoph and Le, 2017, Pham et al., 2018, Liu et al., 2018b]. A *cell* consists of a relatively small section of a neural network, usually 6-12 nodes forming a DAG. A neural architecture is then built by repeatedly stacking one or two different cells on top of each other sequentially, separated by downsampling layers. The layout of cells and downsampling layers is called a *hyper-architecture*, and this is fixed, while the NAS algorithm searches for the best cells. The search space over cells consists of all possible directed acyclic graphs of size 6-12, where each node corresponds to one operation, chosen from a set of 3 to 8 operations. It is also common to set a restriction on the number of total edges or the in-degree of each node [Ying et al., 2019, Liu et al., 2018b]. In this work, we focus on searching over a cell of size 7 with up to 10 edges, where the middle five nodes can be set to **conv1x1**, **conv3x3**, or **max-pool3x3**, consistent with the nasbench search space. See Figure 2 in Appendix B.

Bayesian optimization. Bayesian optimization is a leading technique for hyperparameter tuning. In Bayesian optimization for deep learning, the goal is to find the neural network and/or set of hyperparameters in the search space which lead to the best validation accuracy. Formally, Bayesian optimization seeks to optimize a function $\max_{a \in A} f(a)$, where A is the set of neural architectures and/or hyperparameters, and $f(a)$ denotes the validation accuracy of architecture a after training a fixed dataset for a fixed number of epochs.

In the typical Bayesian optimization setting, t_0 architectures are drawn at random from the search space and trained, making up the initial set. Then the topology of $f(A)$ is modeled using a posterior distribution. Typically a Gausssian process posterior is used; in this work, we use an ensemble of neural networks as a posterior. An acquisition function is used to choose the next neural network to query. Common acquisition functions include expected improvement or upper confidence bounds. We optimize the acquisition function using a mutation algorithm. See Algorithm 1.

Algorithm 1 Neural BayesOpt

Input: Search space A, dataset D, parameters $t_0,\ T,\ c,\ e$, acquisition function ϕ
Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from a and train them on D
Denote $f(a)$ as the validation accuracy of a after training
For t from t_0 to T,

1. Generate a set of c candidate neural architectures from A by drawing $c/2$ architectures at random, and creating $c/2$ mutations of the k architectures a from $\{a_0, \ldots, a_t\}$ with the highest value of $f(a)$
2. Train an ensemble of meta neural networks on $f(a_0), \ldots, f(a_t)$
3. For each candidate architecture a, evaluate the acquisition function $\phi(a)$
4. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$ and evaluate $f(a_{t+1})$

Output: $a^* = \text{argmin}_{a_i} f(a_i)$ and the test accuracy of a^*

Path-based encoding of neural nets. When encoding a neural network (as input to a meta neural network or a NAS algorithm), prior work has used a binary encoding of the adjacency matrix and either a categorical or a one-hot encoding for the operations on each node [Wang et al., 2018, Ying et al., 2019, Deng et al., 2017, Baker et al., 2017]. It is challenging even for a neural network to learn graph topologies directly from an adjacency encoding, and we show in the next section that this method does not perform well. We introduce a path-based encoding and show that it substantially increases the performance of the meta neural network. A path encoding of a cell is created by enumerating all possible paths from the input node to the output node, in terms of the operations (see Figure 2 in Appendix B). The total number of paths is $\sum_{i=0}^{n} q^i$ where n denotes the number of nodes in the cell, and q denotes the size of the set of operations for each node. For example, the nasbench cell search space has $\sum_{i=0}^{5} 3^i = 364$ possible paths.

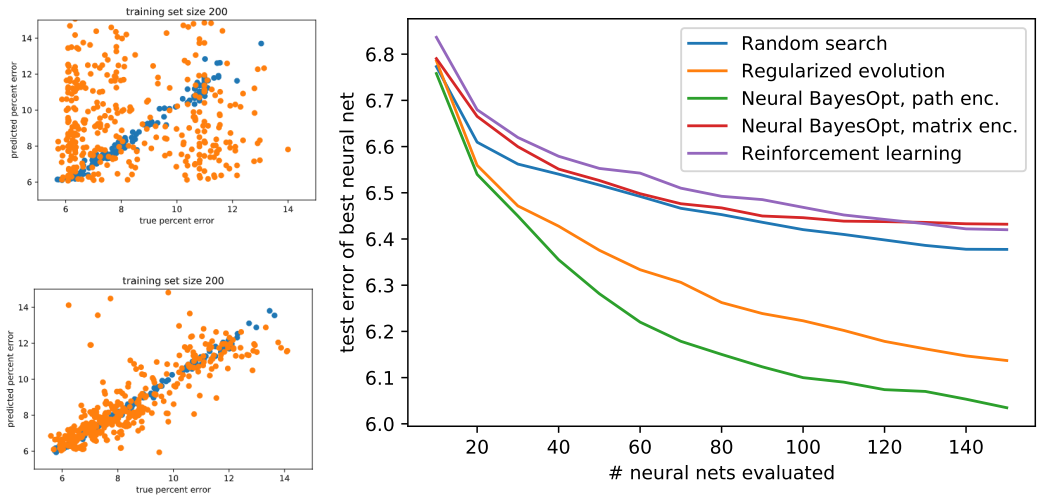

Figure 1: Performance of the meta neural network with an adjacency matrix encoding (top left) and path encoding (bottom left). The training set is in blue, and the test set is in orange. Performance of Neural BayesOpt compared to benchmarks (right).

4 Experiments

In this section, we discuss our experimental setup and results. We give experimental results on the performance of the meta neural network itself, as well as the full NAS algorithm compared to several baselines. We use the nasbench dataset described in the previous section. There are approximately 423,000 unique architectures, and the best architecture achieves 5.68% test error on CIFAR-10.

Meta neural network experiments. We evaluate the performance of the meta neural network. The meta neural network consists of a sequential fully-connected neural network. The number of layers is set to 10, and each layer has width 20. We use the Adam optimizer with a learning rate of 0.1. These were chosen through random hyperparameter search. We set the loss function to mean absolute error. We trained the meta neural network on 200 neural networks, and tested the standard adjacency matrix encoding as well as the path encoding discussed in Section 3. See Figure 1 (left). The path-based encoding outperforms the adjacency matrix encoding by a factor of 3 in mean absolute error.

NAS experiments. We compare our NAS algorithm to random search, regularized evolution, and reinforcement learning. Each NAS algorithm is given a budget of 150 queries. That is, each algorithm can train and output the validation accuracy of at most 150 architectures. Every 10 iterations, we output the architecture with the best validation error found by the algorithm so far. After all NAS algorithms have completed, we return the test error for each architecture outputted. We ran 200 trials for each algorithm. For the Neural BayesOpt algorithm, we used an ensemble of size 5 and UCB.

The Neural BayesOpt algorithm with path encoding significantly outperformed all other baselines, and is state-of-the-art on nasbench for the 100-150 queries setting [Ying et al., 2019, Borsos et al., 2019, Maziarz et al., 2018]. See Figure 1 (right). The standard deviation among 200 trials for all NAS algorithms were between 0.16 and 0.22 (Neural BayesOpt had the lowest standard deviation). In Appendix B, we perform more experiments (comparing five different acquisition functions, giving the NAS algorithms a budget of 500 queries, and comparing different ensemble sizes). We also run through the NAS research checklist [Lindauer and Hutter, 2019] discussing e.g. releasing our code and comparing wall-clock time vs. number of queries.

5 Conclusion

In this work, we propose a novel method for neural architecture search. Our method uses Bayesian optimization with a meta neural network predictive model, and we encode the neural networks using a path-based encoding scheme. This allows the meta neural network to accurately predict the validation

accuracy of new neural networks. Our NAS algorithm significantly outperforms all other baselines. Interesting follow-up questions include testing our method against other algorithms, e.g., AlphaX, testing on other search spaces, e.g., the search space from DARTS [Liu et al., 2018b], and designing a multi-fidelity version of our approach.

References

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

Zalán Borsos, Andrey Khorlin, and Andrea Gesmundo. Transfer nas: Knowledge transfer between search spaces with transformer agents. *arXiv preprint arXiv:1906.08102*, 2019.

Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

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

Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary intelligence*, 1(1):47–62, 2008.

Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Roxana Istrate, Florian Scheidegger, Giovanni Mariani, Dimitrios Nikolopoulos, Costas Bekas, and A Cristiano I Malossi. Tapas: Train-less accuracy predictor for architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 2018.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018.

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

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. *ICLR 2017*, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. *arXiv preprint arXiv:1909.02453*, 2019.

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

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

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Krzysztof Maziarz, Andrey Khorlin, Quentin de Laroussilhe, and Andrea Gesmundo. Evolutionary-neural hybrid agents for architecture search. *arXiv preprint arXiv:1811.09828*, 2018.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

Syed Asif Raza Shah, Wenji Wu, Qiming Lu, Liang Zhang, Sajith Sasidharan, Phil DeMar, Chin Guok, John Macauley, Eric Pouyoul, Jin Kim, et al. Amoebanet: An sdn-enabled network service for big data science. *Journal of Network and Computer Applications*, 119:70–82, 2018.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.

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

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Linnan Wang, Yiyang Zhao, Yuu Jinnai, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1805.07440*, 2018.

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

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

A Related Work Continued

In this section, we extend the related work discussion from section 2.

Neural architecture search Evolutionary algorithms for neural architecture search have been studied since at least the 1990s [Floreano et al., 2008, Kitano, 1990, Stanley and Miikkulainen, 2002]. Since the deep learning revolution in 2012 [Krizhevsky et al., 2012], NASNet was the first neural architecture search algorithm to gain significant attention [Zoph and Le, 2017]. NASNet is a novel reinforcement learning algorithm for NAS which achieves state-of-the-art results on CIFAR-10 and PTB, however, the algorithm requires 3000 GPU days to train. NASNet spurred a number of follow-up work, eventually pushing the required computation for NAS on CIFAR-10 and PTB to a single GPU day [Pham et al., 2018, Liu et al., 2018a,b, Ying et al., 2019, Kandasamy et al., 2018, Jin et al., 2018, Shah et al., 2018]. Some of the most popular techniques for NAS include evolutionary algorithms [Shah et al., 2018], reinforcement learning [Zoph and Le, 2017, Pham et al., 2018, Liu et al., 2018a, Tan and Le, 2019], Bayesian optimization [Kandasamy et al., 2018, Jin et al., 2018], and gradient descent [Liu et al., 2018b].

The choice of search space is an important element in NAS research [Elsken et al., 2018]. Two popular techniques are progressive search spaces [Liu et al., 2018a, Kandasamy et al., 2018, Jin et al., 2018], and cell-based search spaces [Zoph and Le, 2017, Pham et al., 2018, Liu et al., 2018b, Li and Talwalkar, 2019].

Recent papers have shown that random search over cell-based search spaces are on par with the most popular NAS algorithms and call for fair and reproducible experiments in the future [Li and Talwalkar, 2019, Sciuto et al., 2019]. In this vein, the nasbench dataset was created, which contains over 400k neural architectures with precomputed training, validation, and test accuracy [Ying et al., 2019].

A recent algorithm, AlphaX, uses a meta neural network to perform NAS [Wang et al., 2018], which is a similar high-level idea as our work but with many key differences. Their search is progressive, and each iteration makes a small change to the current neural network, rather than choosing a completely new neural network. Therefore, their meta neural network is trained to optimize the selection of new actions as well as predict the performance of the new progressive network. Furthermore, they use an adjacency matrix featurization for the inputs to the meta neural network, which we find to perform worse than our path-based encoding.

Bayesian optimization Bayesian optimization is a leading technique for zero-th order optimization when function queries are expensive [Rasmussen, 2003, Frazier, 2018], and it has seen great success in hyperparameter optimization for deep learning [Rasmussen, 2003, Golovin et al., 2017, Li et al., 2016]. The majority of Bayesian optimization literature has focused on problems where the domain is Euclidean or categorical, and where the prior is a Gaussian process [Rasmussen, 2003, Golovin et al., 2017, Frazier, 2018, Snoek et al., 2012]. More recently, Bayesian optimization with a Gaussian process prior has been used for progressive neural architecture search [Kandasamy et al., 2018, Jin et al., 2018].

Predicting neural network accuracy There are several works which predict the validation accuracy of neural networks. Peephole uses a layer-wise encoding of neural networks with an LSTM algorithm to predict neural net accuracy [Deng et al., 2017], and TAPAS uses a layer-wise encoding and dataset features to predict the accuracy for neural network + dataset pairs [Istrate et al., 2019]. There are several works which predict the learning curve of neural networks for hyperparameter optimization [Klein et al., 2017, Domhan et al., 2015] or NAS [Baker et al., 2017] using Bayesian techniques.

B Experiments Appendix

Nasbench. We describe the nasbench dataset in more detail. For the full details, see [Ying et al., 2019]. This dataset was created to facilitate NAS research, by making it possible to run NAS experiments without a vast amount of compute power, by making it feasible to run enough trials to reach statistical significant results, and by creating a fair testing ground for existing NAS algorithms.

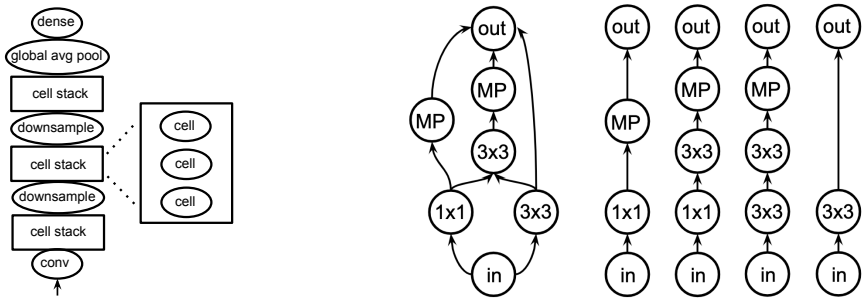

Figure 2: Hyper-architcture of a nasbench neural network (left) Example of a path encoding of a neural network (right).

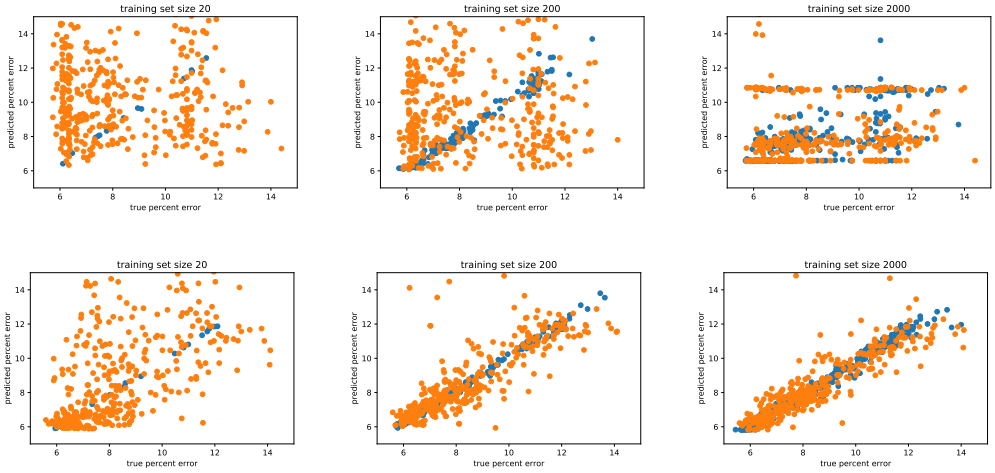

Figure 3: Performance of meta neural network with adjacency matrix encoding (row 1) and path encoding (row 2). The training set is in blue, and the test set is in orange.

The search space consists of a cell with 7 nodes. The first node is the input, and the last node is the output. The remaining five nodes can be either **conv1x1**, **conv3x3**, or **max-pool3x3** operations. The cell can take on any DAG structure from the input to the output with at most 9 edges. The nasbench search space was chosen to contain ResNet-like and Inception-like cells [He et al., 2016, Szegedy et al., 2016]. The hyper-architecture consists of nine cells stacked sequentially, with each set of three cells separated by downsampling layers. The first layer before the first cell is a convolutional layer, and the hyper-architecture ends with a global average pooling layer and dense layer. See Figure 2. There are approximately 423,000 architectures in the nasbench search space, after removing isomorphisms.

Meta neural network experiments. First, we test the effectiveness of the path-based encoding versus the adjancency matrix encoding on different training set sizes. We trained the meta neural network on 20, 200, and 2000 neural networks. We used a test set of size 500. See Figure 3. The path-based encoding outperforms the adjacency matrix encoding by a factor of 3 in mean absolute error. We note that a training set of size 20 neural networks is realistic at the start of a NAS algorithm, and size 200 is realistic near the middle or end of a NAS algorithm.

Next, we tested the effect of the ensemble size. In this model, we train multiple meta neural networks with different random seeds, and then predict based on the mean of all predictions in the ensemble. We trained ensembles of meta neural networks with path-based encoding with a training set of size 200 and a test set of size 500. We tested ensemble size 1, 5, and 25, and we see the test error decreases from 0.995 to 0.911. See Figure 4.

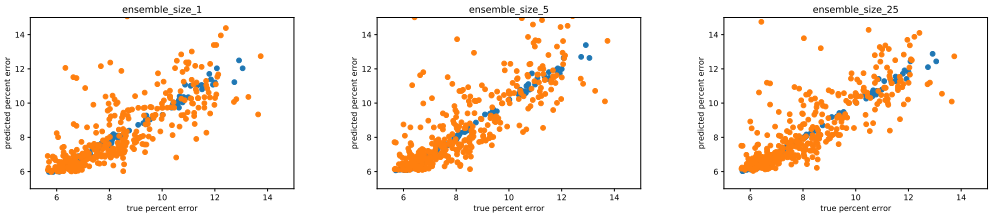

Figure 4: Performance of meta neural network with ensembles of size 1, 5, or 25.

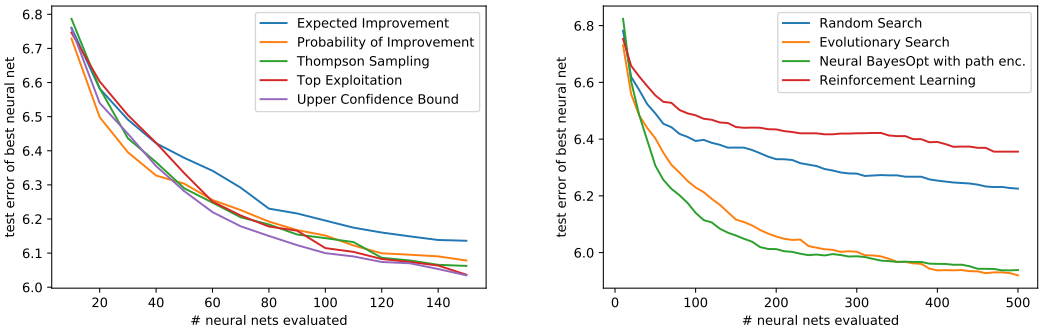

Figure 5: Comparison of different acquisition functions (left). NAS algorithms with a budget of 500 queries (right).

NAS experiments. We give a brief description of each baseline we tested.

Random search (RS) The simplest baseline, random search draws n architectures at random and outputs the architecture with the highest validation accuracy. Despite its simplicity, multiple papers have concluded that random search is a competitive baseline for NAS algorithms [Li and Talwalkar, 2019, Sciuto et al., 2019].

Regularized evolution(RE) We used the nasbench implementation of regularized evolution [Ying et al., 2019]. The algorithm consists of drawing an initial set of cells at random, and then iteratively mutating the best architecture out of a sample of all architectures evaluated so far. The architectures with the worst validation accuracy in each iteration are removed from the population.

Reinforcement Learning (RE) We used the nasbench implementation of reinforcement learning [Ying et al., 2019], based on the REINFORCE algorithm [Williams, 1992]. We chose this algorithm because prior nasbench experiments have shown that a 1-layer LSTM controller trained with PPO is not effective on the nasbench dataset [Ying et al., 2019].

We run three more experiments: testing out different acquisition functions, using a slightly different validation accuracy metric, and running NAS algorithms for 500 queries rather than 150.

Acquisition functions. We ran experiments for five different acquisition functions: expected improvement, probability of improvement, Thompson sampling, max exploitation, and upper confidence bound. The results are shown in Figure 5. We ran 100 trials of each algorithm.

NAS algorithms running for 500 queries. In our main set of experiments, we gave each NAS algorithm a budget of 150 queries. That is, each NAS algorithm can only choose 150 architectures to train and evaluate. Now we give each NAS algorithm a budget of 500 queries. We tested random search, regularized evolution, and Neural BayesOpt with a path encoding, and we ran 100 trials for each algorithm. We see that regularized evolution and Neural BayesOpt both give roughly the same results after query 350. A possible explanation is that both algorithms find the optimal or near-optimal architecture by query 350.

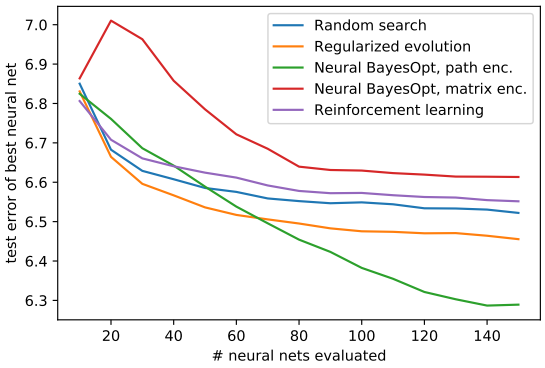

Figure 6: NAS experiments with random validation error and mean test error.

Mean vs. random validation accuracy. In the nasbench dataset, each architecture was trained to 108 epochs three separate times with different random seeds. The nasbench paper conducted experiments by choosing a random validation error when querying each architecture, and then reporting the mean test error at the conclusion of the NAS algorithm. We found that the mismatch (choosing random validation error, but mean test error) added extra noise that would not be present during a real-life NAS experiment. Another way to conduct experiments on nasbench is to use mean validation error and mean test error. This is the method we used in Section 4 and the previous experiments from this section. Perhaps the most realistic experiment would be to use a random validation error and the test error corresponding to that validation error, however, the nasbench dataset does not explicitly give this functionality.

In Figure 6, we give results in the setting of the nasbench paper, using random validation error and mean test error.

Best practices checklist for NAS. Following calls for fair and reproducible NAS research [Li and Talwalkar, 2019, Sciuto et al., 2019, Ying et al., 2019], a best practices checklist was recently created [Lindauer and Hutter, 2019]. We address the points on the checklist. Note that many points are addressed by virtue of using the nasbench datast.

- It is challenging to release our code during our anonymous submission, but we plan to release our code soon for the non-anonymous version of this paper. Our code is already cleaned up.
- Since we used the nasbench dataset, we already check many checkboxes. For example, the training pipeline is already public, the hyperparameters for training are fixed, all of our benchmarks are evaluated in the same way (and we used two different evaluation methods), we ran 200 trials each, and confounding factors are controlled.
- All of the benchmarks we used were taken as-is from the nasbench or nas_benchmarks repository [Ying et al., 2019], and we compared against random search.
- We tested our algorithm with and without the path encoding, and with several acquisition functions, as an ablation study.
- We plotted our results with respect to number of queries, not wall-clock time. Our method is a black-box NAS algorithm, and we only compared against other black-box algorithms. Therefore, number of queries is highly correlated with wall-clock time. For instance, the average time to train a nasbench architecture on CIFAR-10 for 108 epochs is 18.6 minutes, and it takes less than one minute to train and test our meta neural network with 200 datapoints. However, in future work we will plot our results with respect to wall-clock time.