Katib: A Distributed General AutoML Platform Based on Kubernetes

Jinan Zhou
Cisco Systems, USA
jinazhou@cisco.com

Andrey Velichkevich
Cisco Systems, USA
avelichk@cisco.com

Kirill Prosvirov
Cisco Systems, USA
kprosvir@cisco.com

Yuji Oshima
NTT Software Innovation Center, Japan
yuuji.ooshima.fn@hco.ntt.co.jp

Anubhav Garg
Cisco System, USA
anubhgar@cisco.com

Abstract

Automatic Machine Learning (AutoML) has been gaining popularity recently. However, a unified infrastructure to perform AutoML tasks has not yet been developed. To solve this problem, Katib is designed. Katib is a general AutoML platform that runs in distributed systems. Integrated with all the necessary functions, it can support an eclectic range of AutoML algorithms including both hyperparameter tuning and neural architecture search. The system is divided into separate components, all of which are encapsulated as microservices. Each microservice operates inside a Kubernetes pod and communicates with others via well-defined APIs, thus allowing flexible management and scalable deployment at a minimal cost. Together with a powerful user interface, Katib provides a universal platform for researchers as well as enterprises to try, compare and deploy their AutoML algorithms.

1 Introduction

Automatic Machine Learning (AutoML) is the artificial intelligence domain that studies how to search for the optimal hyperparameters or neural network structure for a specific task. It is not only accessible to the user with limited computer science backgrounds, but also can discover state-of-art models that are highly comparable to the ones hand crafted by human experts ([21], [14], [16], [4], [10]). However, several problems need to be solved before AutoML can be put into real production. The first is the diversity of AutoML algorithms. Algorithms for hyperparameter tuning are generally different from those for neural architecture search (NAS). Even within the domain of NAS, different algorithms follow separate mechanisms. This diversity makes it difficult to reuse infrastructure and code, thus increases the cost to deploy AutoML at large scale. The second problem is the prohibitive computational cost. The algorithm proposed by Zoph [23], for example, takes days to run even with hundreds of GPUs.

To solve the first problem, we propose to build a general AutoML system. By summarizing all the AutoML algorithms from a higher perspective, it is feasible to integrate different components needed by different algorithms into one framework. To solve the second problem, we exploited the technology of distributed systems and containers. Our proposed system is divided into functionally independent components. With the help of Kubernetes [1], each component can be encapsulated inside a container as a microservice.

Our contributions can be summarized as follows:

- We integrated various hyperparameter tuning and neural architecture search algorithms into one system.
- We standardized the interface to define and deploy AutoML workflows in distributed systems.

The implementation of Katib is available at `https://github.com/kubeflow/katib`.

2 AutoML Workflows

AutoML algorithms share the common ground that they run in an iterative manner. The user first defines the search space, metrics target and maximum iterations. The algorithm searches for the optimal solution until the target metrics or maximum number of iterations is reached. However, they may vary in terms of their internal mechanisms.

2.1 Hyperparameter Tuning

In hyperparameter tuning, we have a black-box function $f(\cdot)$ whose value is dependent on a set of parameters p. The goal is to find a $\hat{p}$ such that $f(p)$ can be minimized or maximized. In each iteration, a search algorithm service `Suggestion` will generate a set of candidate hyperparameters. The candidates are sent to `Trial` that provides training and validation services. The performance metrics are then collected by `Suggestion` to improve its next generation.

2.2 Neural Architecture Search

In neural architecture search (NAS), a neural network is represented by a directed acyclic graph (DAG) $G = \{V, E\}$, where a vertex V_i denotes the latent representation in i^{th} layer and a directed edge $E_k = (V_i, V_j)$ denotes an operation o_k whose input is V_i and output is given to V_j. The value of a vertex V_i depends on all the incoming edges:

$$V_i = g(\{o_k(V_j) | (V_i, V_j) \in E\})$$

$g(\cdot)$ is a function to combine all the inputs. It can vary in different algorithms. In [23] and [16], $g(\cdot)$ means concatenating along the depth dimension. In [4], [21] and [14], a weight is assigned to every edge so $g(\cdot)$ is naturally the weighted sum.

Extensive research in NAS has lead to enormous diversity of NAS solutions. In terms of searching objective, the algo-

rithm may want to search for either the optimal network or the optimal cell. The former constructs the whole graph directly while the latter generates a subgraph G', and the whole graph is built by duplicating the topology of G'. In terms of evolving strategy, some NAS algorithms adopt a generation approach while the others use modification. In generation approach, the algorithm will propose a new neural architecture in each iteration. In modification approach, however, the algorithm will modify the current architecture by adding or deleting some parts of it instead of creating a brand-new candidate. Based on this categorization, the latest NAS algorithms can be summarized as follows:

Strategy	Search for Network	Search for Cell
Evolve by Generation	[23], [16], [20], [11], [2]	[23], [16], [24], [22], [13]
Evolve by Modification	[9], [4], [18], [7], [3]	[17], [21], [14], [10], [15], [6], [5]

Table 1: Summary of neural architecture search algorithms

Diverse as they are, those algorithms can be integrated into one system. Compared with hyperparameter tuning, NAS only needs one extra `ModelManager` service to store, construct and manipulate models. In each iteration, `Suggestion` provides the topology of the next candidate or the modification decisions of the previous architecture to `ModelManager`, which constructs the model and sends it to `Trial`. Then the model is evaluated and the performance metrics are fed back to `Suggestion`, starting a new iteration.

All these workflows can be summarized by Figure 1:

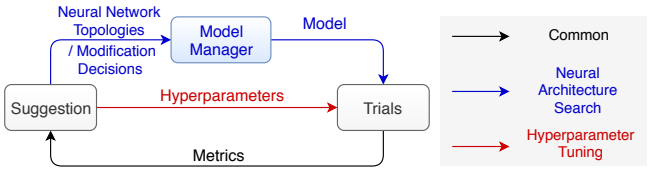

Figure 1: Summary of AutoML workflows

3 Katib System Design

Combining the requirements of these AutoML workflows and the ideas from Google's black-box optimization tool Vizier [8], we come up with design in Figure 2. The user starts from defining an AutoML task with Katib's interface, the details of which can be found at `http://bit.ly/2E5B9pV`. A controller examines the task definition and spawns the necessary services. The data communication between different containers is managed by Vizier Core. The searching procedure follows exactly the workflow defined in Section 2.

We shall take EnvelopeNet [10] as an example to show how Katib can support complicated AutoML algorithms with non-standard schemes. In EnvelopeNet, the neural networks are updated by pruning and expanding *EnvelopeCells*, which are convolution blocks connected in parallel. And these modification decisions are based on *feature statistics* instead

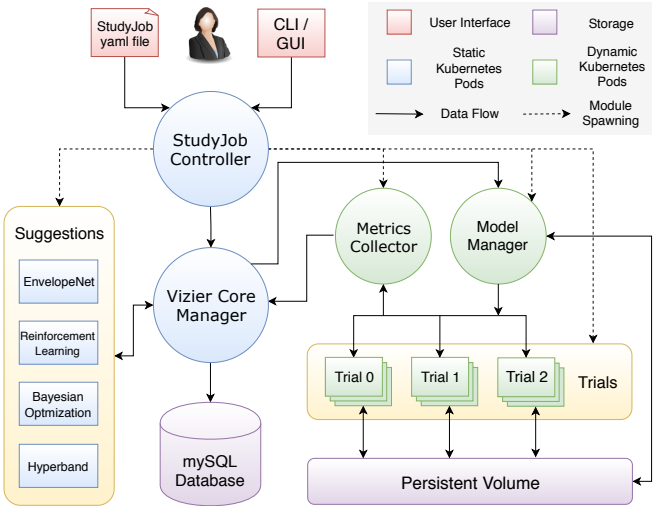

Figure 2: Design of Katib as a general AutoML system

of validation accuracy. The `Suggestion` and training containers will be pre-built so the user only needs to specify the structure of *EnvelopeCells* and other necessary parameters in StudyJob yaml file. In each iteration, Vizier Core Manager first requests one or more modification decisions from `Suggestion` and sends them to `ModelManager`. Then `ModelManager` calculates the current architectures, compiles the models into a runnable objects, and sends them to `Trial`, which carries out a truncated training process. Once finished, a `MetricsCollector` is spawned to parse *feature statistics* from the training logs. Finally, this information is fed back to `Suggestion` via Vizier Core and a new iteration starts. During the process, all the model topologies and metrics are stored in a database and presented to the user.

Katib is built to scale. Upon request, the controller can spawn multiple parallel `Trials` in each iteration to accelerate the search. What's more, the `Suggestion`, controller and Vizier Core service can be shared among all the users.

The initial version provides hyperparameter tuning with Bayesian optimization [19], Hyperband [12], grid search and neural architecture search with reinforcement learning ([23]). Apart from using these provisioned components, the user can also deploy customized tasks by creating his own algorithm for `Suggestion` and training container for `Trial`. In the near future, we will add more algorithms such as EnvelopeNet [10] and integrate the support for advanced acceleration techniques such as parameter sharing [16].

4 Conclusion

This paper presents Katib, a distributed general AutoML system based on Kubernetes. The key idea is to abstract AutoML algorithms into the interaction between functionally isolated components and containerize each component as a microservice. With this extendable and scalable design, Katib can be a powerful tool for both advancing machine learning research and delivering turnkey AI solutions to customers.

References

[1] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, (3):81–84, 2014.

[2] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[3] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. AAAI, 2018.

[4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[5] Yukang Chen, Qian Zhang, Chang Huang, Lisen Mu, Gaofeng Meng, and Xinggang Wang. Reinforced evolutionary neural architecture search. *arXiv preprint arXiv:1808.00193*, 2018.

[6] Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 874–883. JMLR. org, 2017.

[7] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

[8] 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.

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

[10] Purushotham Kamath, Abhishek Singh, and Debo Dutta. Neural architecture construction using envelopenets. *arXiv preprint arXiv:1803.06744*, 2018.

[11] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191*, 2018.

[12] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

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

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

[15] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, pages 7827–7838, 2018.

[16] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, 2018.

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

[18] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[19] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[20] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[21] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[22] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2423–2432, 2018.

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

[24] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.