Journal of Machine Learning Research 24 (2023) 1-6 Submitted 11/20; Revised 12/22; Published 1/23

AutoKeras: An AutoML Library for Deep Learning

Haifeng Jin [1, 2] HAIFENGJ@GOOGLE.COM
François Chollet [1] FCHOLLET@GOOGLE.COM
Qingquan Song [2] USTCSQQ@GMAIL.COM
Xia Hu [3] XIA.HU@RICE.EDU
[1] *Google LLC, Mountain View, CA 94043, USA*
[2] *Texas A&M University, College Station, TX 77843, USA*
[3] *Rice University, Houston, TX 77005, USA*

Editor: Andreas Mueller

Abstract

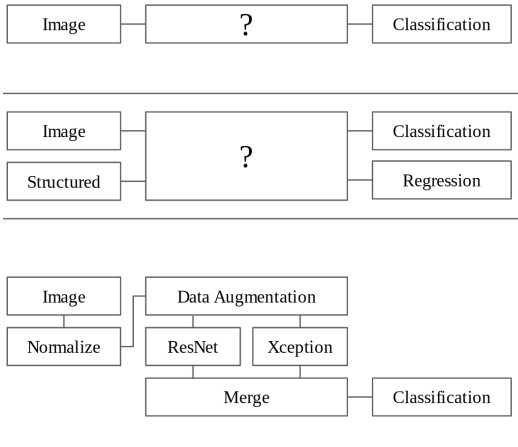
To use deep learning, one needs to be familiar with various software tools like TensorFlow or Keras, as well as various model architecture and optimization best practices. Despite recent progress in software usability, deep learning remains a highly specialized occupation. To enable people with limited machine learning and programming experience to adopt deep learning, we developed AutoKeras, an Automated Machine Learning (AutoML) library that automates the process of model selection and hyperparameter tuning. AutoKeras encapsulates the complex process of building and training deep neural networks into a very simple and accessible interface, which enables novice users to solve standard machine learning problems with a few lines of code. Designed with practical applications in mind, AutoKeras is built on top of Keras and TensorFlow, and all AutoKeras-created models can be easily exported and deployed with the help of the TensorFlow ecosystem tooling.

Keywords: AutoML, Machine Learning, Deep Learning, Python

1. Introduction

Deep learning has been widely adopted for its success in many real-world applications like computer vision (He et al., 2016) and natural language processing (Devlin et al., 2019). To adopt deep learning, people often need to go through a non-trivial learning curve (Bargava, 2018; Song et al., 2022). A strong foundation of machine learning theory and being proficient in deep learning libraries like TensorFlow (Abadi et al., 2016) or Keras (Chollet et al., 2015) are both prerequisites for building a deep learning solution (Yao et al., 2018).

To remove the barriers to adopting deep learning, we developed AutoKeras, an AutoML library for deep learning. It automates the process of model selection and hyperparameter tuning and encapsulates the end-to-end process from raw datasets to trained machine learning models into an extremely simple and flexible interface. Novice users can implement deep learning models with a few lines of code, while advanced users can also easily customize different parts of the model to their needs. AutoKeras specializes in raw data types like images and texts in addition to structured data, which is supported by existing AutoML libraries (Thornton et al., 2013; Feurer et al., 2015; Olson et al., 2016; Kotthoff et al., 2017; Feurer et al., 2020; Erickson et al., 2020; Zimmer et al., 2021). It is also flexible enough to cover multi-modal data and multi-task use cases. AutoKeras is built base on




```
1  import autokeras as ak
2  # Task API
3  auto_model = ak.ImageClassifier()
4  auto_model.fit(x_train, y_train)
5  result = auto_model.predict(x_test)
6  # Input/Output API
7  auto_model = ak.AutoModel(
8    inputs=[ak.ImageInput(), ak.StructuredDataInput()],
9    outputs=[ak.ClassificationHead(), ak.RegressionHead()])
10 auto_model.fit(x=[x_image, x_struct], y=[y_clf, y_reg])
11 # Functional API
12 input_node = ak.ImageInput()
13 output_node = ak.Normalization()(input_node)
14 output_node = ak.ImageAugmentation()(output_node)
15 output_node_1 = ak.ResNetBlock(version='v2')(output_node)
16 output_node_2 = ak.XceptionBlock()(output_node)
17 output_node = ak.Merge()([output_node_1, output_node_2])
18 output_node = ak.ClassificationHead()(output_node)
19 auto_model = ak.AutoModel(input_node, output_node)
20 auto_model.fit(x_train, y_train)
```

Figure 1: Three Levels of APIs

KerasTuner (O'Malley et al., 2019), Keras (Chollet et al., 2015), and TensorFlow (Abadi et al., 2016). The models created by AutoKeras can be easily exported as Keras models, which can be deployed in various production environments with the help of the TensorFlow ecosystem.

2. API Design

The API design of AutoKeras follows the style of Keras, which is well-received by the deep learning community. It has three levels of APIs, namely, task API, IO API, and functional API, ranging from the simplest to the most configurable. The code for using these APIs is shown in Figure 1 with diagrams showing the corresponding neural network models. The parts with question marks are tuned automatically.

The task API requires the least amount of configurations from the user. As shown in Figure 1 from line 3 to 5, an example of the image classification task is implemented within three lines of code. Six different tasks are supported in task APIs, including classification and regression for image, text, and structured data.

The IO API (input/output API) supports multi-modal data and multi-task use cases. In Figure 1 from line 7 to 10, the dataset is a set of images with attributes, for example, an image of a house with attributes describing the total area and location of the house. Each data sample is associated with two prediction targets, a label for classification, and a real value for regression. The user needs to specify the inputs and outputs format of the model as shown in line 8 and 9. The training data are passed in lists in the same order in line 10.

The functional API enables advanced users to tailor the search spaces according to their needs. It resembles the Keras functional API to let the user build the computational graph of the deep learning model with the building blocks. The example from line 12 to line 19 connects both preprocessing steps and neural network blocks, which apply data normalization and data augmentation to the data before passing it to a neural network with ResNet (He et al., 2016) and XceptionNet (Chollet, 2017). Notably, on line 15, the

version of the ResNet is specified as v2, which further reduces the size of the search space. There are many such configurable hyperparameters for other blocks as well. They are tuned automatically if left unspecified. Moreover, the users can also create custom neural network blocks to use with the functional API.

Compared with other AutoML libraries, like AutoGluon (Erickson et al., 2020), which covers tree-based models in the search space, and Auto-PyTorch (Zimmer et al., 2021), which focus on structured data tasks, AutoKeras is optimized for raw data types and focuses on deep neural network models only, which makes it fully compatible with the TensorFlow and Keras ecosystem. The fit function in AutoKeras supports all the arguments supported by the Keras fit function. The model found by AutoKeras can be easily exported as a Keras model. With the help of the TensorFlow ecosystem, it is ready for deployment in various production environments.

3. System Architecture

AutoKeras uses Keras and TensorFlow to build machine learning models. KerasTuner, a hyperparameter tuning framework for Keras, provides the infrastructure for implementing the search space and the search algorithm. Built on top of KerasTuner, AutoKeras implements a series of carefully designed search spaces, task-specific search algorithms, and easy-to-use APIs.

The core AutoKeras workflow consists of the following steps. First, AutoKeras analyzes the training data to determine e.g. whether a given tabular data feature is categorical or numerical, whether the image data includes a channel dimension, or whether the classification labels need to be encoded. Second, it uses this information to construct a suitable search space that encompasses both neural architecture patterns and common hyperparameters. Finally, the search algorithm finds high-performing hyperparameter values.

The search space of AutoKeras includes state-of-the-art deep learning models for the supported tasks. For models like EfficientNet (Tan and Le, 2019) and BERT (Devlin et al., 2019), pretrained weights can be leveraged. Besides optimizing the model architecture, it also tunes the hyperparameters from the preprocessing steps and the training process, for example, image data augmentation, text vectorization, categorical feature encoding, optimizer, learning rate, and weight decay.

4. Search Algorithm

Instead of treating hyperparameter tuning as a black-box optimization problem, AutoKeras implements a novel search algorithm that leverages the prior knowledge of the search space. The main idea is to warm-start the search with good configurations (a configuration is a complete set of hyperparameter values that builds and trains a model) and to keep exploiting the neighborhood of good configurations.

Under the task API of AutoKeras, the search space is predefined. Instead of starting from random configurations, it starts by evaluating a list of predefined configurations, which are known to perform well generally. Then, the search algorithm will always mutate the current best configuration to create the next configuration to evaluate. Such design is

inspired by the hill-climbing algorithm (Elsken et al., 2018). The pseudo-code of the search algorithm is shown in Algorithm 1.

Algorithm 1 The search algorithm of AutoKeras

```
for i ← 1 to t do                          ▷ t is the total number of evaluations in the search
    if i <= m then                         ▷ m is the number of predefined configs
        eval(ith pre-defined hp)           ▷ Evaluate pre-defined configurations
    else
        eval(mutate(get_best_hp()))        ▷ Mutate the current best for evaluation
```

In the mutation process, prior knowledge of the search space is used again. The hyperparameters are hierarchically grouped into sub-modules according to their locations in the model. A sub-module can be a single hyperparameter, a layer, or the entire model. To make the mutated configuration similar to the current best, in every mutation, only one of the sub-modules is selected and all of its hyperparameter values are resampled. To make sub-modules with more hyperparameters less likely to be selected, we assign probabilities for the sub-modules to be selected as follows. A raw probability vector $\hat{\boldsymbol{p}}$ is defined as:

$$\hat{\boldsymbol{p}} = (\frac{1}{n_1 + 1}, \frac{1}{n_2 + 1}, \ldots, \frac{1}{n_K + 1}) \in \mathbb{R}^K, \tag{1}$$

where n_i is the number of hyperparameters in the ith sub-module, K is the total number of sub-modules, the $+1$ offset is to smooth the small values. To normalize $\hat{\boldsymbol{p}}$ to sum to one, the $\text{logit}(\cdot)$ function and the softmax function $\sigma(\cdot)$ are applied:

$$\boldsymbol{p} = \sigma(\text{logit}(\hat{\boldsymbol{p}})) = \sigma(-\ln \boldsymbol{n}) \text{ for } \boldsymbol{n} = (n_1, n_2, \ldots, n_K) \in \mathbb{R}^K,\ \text{logit}(x) = \ln \frac{x}{1-x}. \tag{2}$$

The normalized vector $\boldsymbol{p}$ contains the final probabilities for the sub-modules to be selected. The experimental results are published on the AutoKeras official website (`autokeras.com`).

5. Conclusions and Future Work

We developed AutoKeras, an AutoML library for deep learning with simple APIs to efficiently provide end-to-end deep learning solutions to the users. It can handle multi-task learning and multi-modal data. The search space is fully customizable by the user. The model found by AutoKeras can be easily exported and deployed in the production environment with the help of the TensorFlow ecosystem.

In the future, we plan to support more tasks, better support distributed search and training to scale up the framework for larger datasets, and build infrastructure to automatically benchmark AutoKeras performance for new releases following the established best practices for neural architecture search (Lindauer and Hutter, 2020).

Acknowledgments

We thank the reviewers for their helpful comments, and we thank all the contributors from our open-source community for their work. This work is, in part, supported by DARPA (#FA8750-17- 2-0116) and NSF (#IIS-1718840 and #IIS-1750074).

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.

Bargava. How to learn deep learning in 6 months. `https://towardsdatascience.com/how-to-learn-deep-learning-in-6-months-e45e40ef7d48`, 2018.

François Chollet et al. Keras. `https://keras.io`, 2015.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. In *ICLR Workshop Track*, 2018.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. AutoGluon-Tabular: Robust and accurate automl for structured data. `arXiv:2003.06505 [stat.ML]`, 2020.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *NeurIPS*, 2015.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. `arXiv:2007.04074 [cs.LG]`, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Lars Kotthoff, Chris Thornton, Holger Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *JMLR*, 2017.

Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. *JMLR*, 2020.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *GECCO*, 2016.

Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Keras Tuner. `https://github.com/keras-team/keras-tuner`, 2019.

Qingquan Song, Haifeng Jin, and Xia Hu. *Automated Machine Learning in Action*. Manning Publications, 2022.

Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICLR*, 2019.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *KDD*, 2013.

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. Taking human out of learning applications: A survey on automated machine learning. `arXiv:1810.13306 [cs.AI]`, 2018.

Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *TPAMI*, 2021.