European Scholar Journal (ESJ)
Available Online at: https://www.scholarzest.com
Vol. 2 No. 6, June 2021,
ISSN: 2660-5562

NORD: USING THE PYTHON PROGRAMMING LANGUAGE TO NEURAL ARCHITECTURE SEARCH

Ko'paysinov Shavkat Sharofiddin o'g'li,
Tashkent Institute of Architecture and Civil Engineering, assistant,
Rakhimov Nodir Ergashovich,
Tashkent Institute of Architecture and Civil Engineering, Senior lecture, Tashkent, Uzbekiston

Article history:

Received: 10th May 2021
Accepted: 22th May 2021
Published: 18th June 2021

Abstract:

Neural architecture search (NAS) is a technique for automating the design of artificial neural networks (ANN), a widely used model in the field of machine learning. NAS has been used to design networks that are on par or outperform hand designed architectures. As manually designing the architectures is quite laborious and challenging to execute without adequate experience, NAS enables discovering novel, state-of-the-art architectures. Nonetheless, successfully implementing NAS processes also requires extensive experience with both neural networks and optimization processes. Methods for NAS can be categorized according to the search space, search strategy and performance estimation strategy used. Neural Operations Research and Development (NORD) decouples implementing and designing the networks, enabling the application of existing methods on novel datasets and fairly comparing results. Thus, it aims to make NAS more accessible to researchers, as well as industry practitioners.

Keywords: Neural Architecture Search, Reinforcement learning, Deep learning, Neural networks.

INTRODUCTION

As deep learning methods have been consistently achieving state of the art performance in various computational tasks previously regarded as exceptionally hard, the role of suitable neural architectures has become apparent [1–4]. Traditionally designed by human experts, neural architectures are becoming increasingly complicated, while adequate knowledge regarding each building block such as layers, regularization, and connectivity is required. Furthermore, different domains, such as image, video, and text, demand entirely different approaches. Neural Architecture Search (NAS) aims to automate designing performant architectures, thus enabling researchers to focus on higher-level tasks, such as designing novel building blocks [5–8].

Neural Operations Research and Development (NORD) is a Python framework utilizing PyTorch as a deep learning back-end that aims to define and implement a NAS pipeline while enabling a unified realization of network descriptions, making comparisons between different NAS methods fair. Furthermore, it aims to separate methodological from network implementations, enabling researchers with less experience in coding deep learning models to study the field of NAS.

PIPELINE DESIGN, IMPLEMENTATION, AND REPRODUCIBILITY

NORD implements a NAS pipeline consisting of three main components; design, implementation, and evaluation of proposed architectures 1. Each of these components is dependent on the other two. The design component concerns algorithms utilized in order to generate architecture specifications. The implementation component is responsible for compiling architecture specifications into fully functioning neural networks.

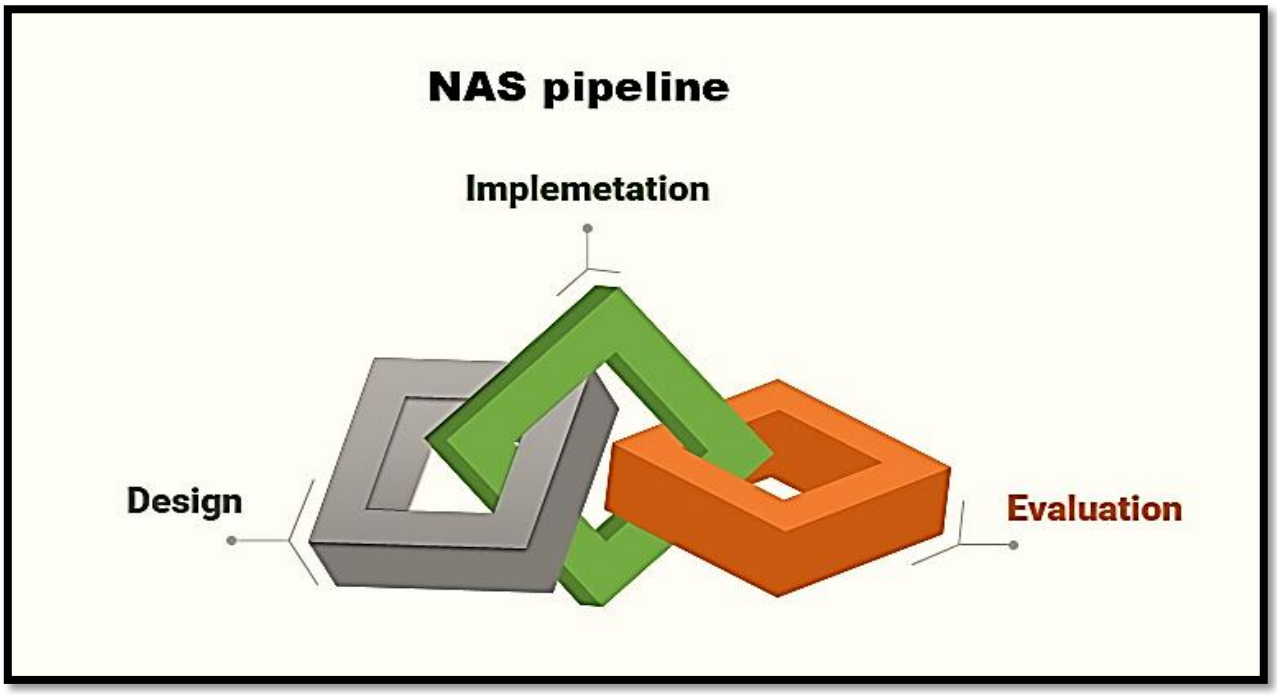

Fig. 1. Basic NAS pipeline.

Finally, the evaluation component is responsible forevaluating the proposed neural networks' quality and providing it to the design algorithm as feedback. While studying the design component is the primary concern of NAS researchers, the other two components can significantly influence experimental outcomes. For example, when implementing a specification that contains multiple inputs ateach layer, how these inputs will be treated can impact the realized network's behavior. Furthermore, the problem of incompatible input shapes arises. Although the design algorithm could account for this problem by designing networks while ensuring that all layers' inputs are compatible, this requires considerably more involvement than we would like from a high-level design algorithm. Finally, the design algorithm should be agnostic to the dataset utilized to evaluate the generated networks' performance, as the same high-level architecture can be successfully applied to various datasets (for example, images with a different number of channels).

We implement data curators, neural descriptors, neural builders, and neural evaluators, to enable agnostic evaluation of design algorithms on various data. Currently, the CIFAR10, Fashion-MNIST, and Activity Recognition from Single Chest-Mounted Accelerometer datasets are implemented, along with the NASBench-101 benchmark dataset of pre-evaluated cell-search architectures. We aim to enrich the available datasets in the near future.

Neural descriptors are containers of network specifications. They treat networks as graphs, with featured nodes. When a new node is added, its base class, instantiation parameters, and name are required. When adding layers to the descriptor with add_layer_sequential the layer's name is not a required parameter, as the layer is automatically connected to the last added layer. Contrary, when adding a layer through add_layer, its name is a required parameter, as the connect_layers function will later be utilized to connect layers by their names.

Neural builders can implement the core architecture that neural descriptors contain while also implementing custom-specified fully connected modules for the network's final layers. Neural builders are also responsible for ensuring that the specification produces a viable, working network to the extent that this is possible. For example, specifications that do not have a path from the input to the output layer do not generate working networks at the moment. Regarding the efficiency and time required to implement a specification, it is negligible compared to the time required to evaluate it. For example, a 10-layer convolutional network with 8M parameters requires 0.11 s to be implemented, while evaluating on the Fashion MNIST [9] dataset requires 43 s on a NVIDIA GTX1060 graphics card for each epoch.

Neural evaluators train and evaluate the generated networks, utilizing the specified dataset, optimization algorithm, loss function, and performance metrics. As a primary concern in NAS literature is the reproducibility of various implementations [10], we provide a simple utility function assure_reproducibility, which seeds the random number generators of numpy and PyTorch, while also forcing deterministic execution of CUDA calls. Although potentially hindering performance, it ensures that any experiment is fully reproducible. Furthermore, visual analysis tools are provided, such as network specification and realization plots with fixed layouts, enabling ease of comparison between various produced architectures. In Fig. 2 we depict the implementation and interactions between the most important NORD modules implementing the NAS pipeline.

Executing NAS on custom datasets is relatively straightforward. After specifying a data loading function, various dataset-related options can be added to the singleton class Configs, such as the data shape, the number of channels, desired loss and metrics, as well as the data loading function itself. One such example exists in the custom_dataset_example.py script, where the MNIST dataset is added to the Configs class.

REINFORCEMENT LEARNING

Reinforcement learning (RL) can underpin a NAS search strategy. Zoph et al. applied NAS with RL targeting the CIFAR-10 dataset and achieved a network architecture that rivals the best manually designed architecture for accuracy, with an error rate of 3.65, 0.09 percent better and 1.05x faster than a related hand-designed model. On the Penn Treebank dataset, that model composed a recurrent cell that outperforms LSTM, reaching a test set perplexity of 62.4, or 3.6 perplexity better than the prior leading system. On the PTB character language modeling task it achieved bits per character of 1.214.

Learning a model architecture directly on a large dataset can be a lengthy process. NASNet addressed this issue by transferring a building block designed for a small dataset to a larger dataset. The design was constrained to use two types of convolutional cells to return feature maps that serve two main functions when convoluting an input feature map: normal cells that return maps of the same extent (height and width) and reduction cells in which the returned feature map height and width is reduced by a factor of two. For the reduction cell, the initial operation applied to the cell's inputs uses a stride of two (to reduce the height and width). The learned aspect of the design included elements such as which lower layer(s) each higher layer took as input, the transformations applied at that layer and to merge multiple outputs at each layer. In the studied example, the best convolutional layer (or "cell") was designed for the CIFAR-10 dataset and then applied to the ImageNet dataset by stacking copies of this cell, each with its own parameters. The approach yielded accuracy of 82.7% top-1 and 96.2% top-5. This exceeded the best human-invented architectures at a cost of 9 billion fewer FLOPS a reduction of 28%. The system continued to exceed the manually-designed alternative at varying computation levels. The image features learned from image classification can be transferred to other computer vision problems. E.g., for object detection, the learned cells integrated with the Faster-RCNN framework improved performance by 4.0% on the COCO dataset.

PUBLICATIONS AND IMPACT

NORD has enabled the investigation of various NAS components, such as the feasibility of utilizing proxy tasks to quickly evaluate network performance and retain relative rankings between architectures [11,12], and the ability of proposed methods to discover adequate macro-architectures [13]. The results indicated that given proxy search spaces of sufficient correlation to the original search space, evolution- ary NAS methods can produce acceptable results, with a up to 10 times speed-up. It has been successfully utilized in HPC environments with both MPI integration (for distributed design algorithms) as well as with Horovod (for distributed network evaluation) under the project DNAD of GRNET.

As NORD strives to make NAS procedures fully modular, the ablative study of individual NAS components is possible. This modularity can significantly impact both the study of novel design methodologies as well as the behavior of existing methods under different conditions. For example, the performance of a global search space strategy applied to a cell search space [14], when different rules are utilized to realize the network specification generated, or when a dataset from a different domain is employed to evaluate the generated architectures. As such, NORD can help to further both research-oriented as well as commercially-oriented utilization of NAS. This is achieved by providing interested parties with essential tools to conduct Neural Architecture Search while also being able to customize any module of the pipeline to suit their needs, such as the design algorithm, dataset, or network realization.

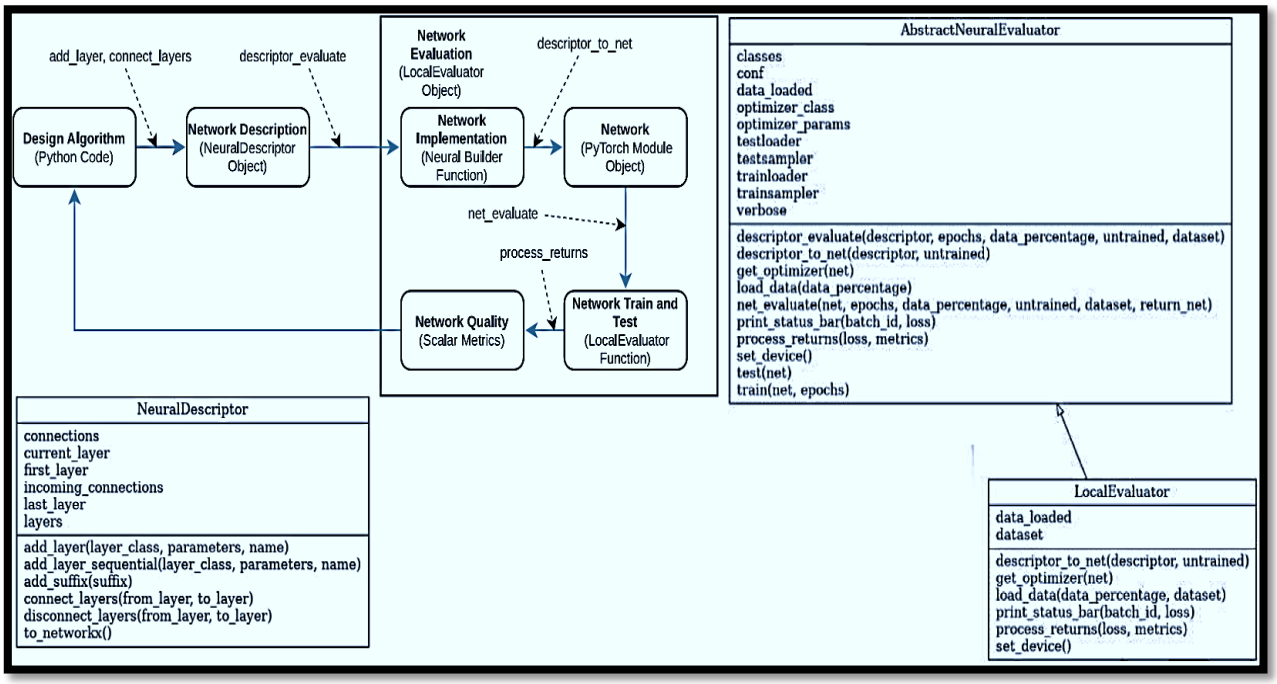

Fig. 2. Nord implementation of NAS pipeline.

```
import torch.nn as nn
import torch.optim as opt
from nord.neural_nets import LocalEvaluator, NeuralDescriptor
from nord.configurations.all import Configs

dataset = 'cifar10'
conf = Configs()
d = NeuralDescriptor()
d.add_layer(nn.Conv2d, {'in_channels': conf.CHANNELS[dataset],
                        'out_channels': 5, 'kernel_size': 3}, 'conv')
d.add_layer(nn.MaxPool2d, {'kernel_size': 2, 'stride': 2}, 'pool')
d.connect_layers('conv', 'pool')

evaluator = LocalEvaluator(optimizer_class=opt.Adam,
                           optimizer_params={}, verbose=True)

loss, metrics, total_time = evaluator.descriptor_evaluate(
    descriptor=d, epochs=2, dataset=dataset)
```

Fig. 3. Defining and evaluating a simple network on CIFAR-10.

CONCLUSIONS AND FUTURE UPDATES

In this work, we present NORD, a python framework for Neural Architecture Search, which aims to simplify the process of designing NAS pipelines. Through NORD, practitioners can implement and compare NAS methods on benchmarks, as well as custom datasets. In summary, NORD enables: the application of existing methodologies on various datasets, the fair comparison of different methods, the ablative study of NAS components, the behavioral analysis of different methodological approaches. As further developments, we aim to implement various design methodologies, application datasets, as well as post-analysis tools, in order to facilitate the comparison of various methods.

DECLARATION OF COMPETING INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

ACKNOWLEDGMENTS

This work was supported by computational time granted from the Greek Research & Technology Network (GRNET) in the National HPC facility ARIS under project ID DNAD, which enabled the development of the method.

The research work was supported by the Hellenic Foundation for Re- search and Innovation (HFRI), Greece under the HFRI Ph.D. Fellowship grant (Fellowship Number: 646).

REFERENCES

1. Elsken, Thomas; Metzen, Jan Hendrik; Hutter, Frank (August 8, 2019). "Neural Architecture Search: A Survey". Journal of Machine Learning Research. 20 (55): 1–21. arXiv:1808.05377. Bibcode:2018arXiv180805377E – via jmlr.org.
2. Wistuba, Martin; Rawat, Ambrish; Pedapati, Tejaswini (2019-05-04). "A Survey on Neural Architecture Search". arXiv:1905.01392 [cs.LG].
3. Jump up to:a b c Zoph, Barret; Le, Quoc V. (2016-11-04). "Neural Architecture Search with Reinforcement Learning". arXiv:1611.01578 [cs.LG].
4. Jump up to:a b c d e Zoph, Barret; Vasudevan, Vijay; Shlens, Jonathon; Le, Quoc V. (2017-07-21). "Learning Transferable Architectures for Scalable Image Recognition". arXiv:1707.07012[cs.CV].
5. He, X., Zhao, K., & Chu, X (2021-01-05). "AutoML: A survey of the state-of-the-art". Knowledge-Based Systems. 212: 106622. arXiv:1908.00709. doi:10.1016/j.knosys.2020.106622. ISSN 0950-7051.
6. 4. Zoph, Barret; Vasudevan, Vijay; Shlens, Jonathon; Le, Quoc V. (November 2, 2017)."AutoML for large scale image classification and object detection". Research Blog. Retrieved 2018-02-20.
7. Hieu, Pham; Y., Guan, Melody; Barret, Zoph; V., Le, Quoc; Jeff, Dean (2018-02-09). "Efficient Neural Architecture Search via Parameter Sharing". arXiv:1802.03268[cs.LG].
8. Real, Esteban; Moore, Sherry; Selle, Andrew; Saxena, Saurabh; Suematsu, Yutaka Leon; Tan, Jie; Le, Quoc; Kurakin, Alex (2017-03-03). "Large-Scale Evolution of Image Classifiers". arXiv:1703.01041 [cs.NE].
9. Jump up to:a b Real, Esteban; Aggarwal, Alok; Huang, Yanping; Le, Quoc V. (2018-02-05). "Regularized Evolution for Image Classifier Architecture Search". arXiv:1802.01548[cs.NE].
10. Stanley, Kenneth; Miikkulainen, Risto, "Evolving Neural Networks through Augmenting Topologies", in: Evolutionary Computation, 2002
11. Thomas, Elsken; Jan Hendrik, Metzen; Frank, Hutter (2017-11-13). "Simple And Efficient Architecture Search for Convolutional Neural Networks". arXiv:1711.04528 [stat.ML].
12. Jump up to:a b Elsken, Thomas; Metzen, Jan Hendrik; Hutter, Frank (2018-04-24). "Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution". arXiv:1804.09081[stat.ML].
13. Jump up to:a b Zhou, Yanqi; Diamos, Gregory. "Neural Architect: A Multi-objective Neural Architecture Search with Performance Prediction" (PDF). Baidu. Retrieved 2019-09-27.
14. Tan, Mingxing; Chen, Bo; Pang, Ruoming; Vasudevan, Vijay; Sandler, Mark; Howard, Andrew; Le, Quoc V. (2018). "MnasNet: Platform-Aware Neural Architecture Search for Mobile". arXiv:1807.11626 [cs.CV].
15. Howard, Andrew; Sandler, Mark; Chu, Grace; Chen, Liang-Chieh; Chen, Bo; Tan, Mingxing; Wang, Weijun; Zhu, Yukun; Pang, Ruoming; Vasudevan, Vijay; Le, Quoc V.; Adam, Hartwig (2019-05-06). "Searching for MobileNetV3". arXiv:1905.02244 [cs.CV].
16. Wu, Bichen; Dai, Xiaoliang; Zhang, Peizhao; Wang, Yanghan; Sun, Fei; Wu, Yiming; Tian, Yuandong; Vajda, Peter; Jia, Yangqing; Keutzer, Kurt (24 May 2019). "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search".arXiv:1812.03443 [cs.CV].
17. Sandler, Mark; Howard, Andrew; Zhu, Menglong; Zhmoginov, Andrey; Chen, Liang-Chieh (2018). "MobileNetV2: Inverted Residuals and Linear Bottlenecks". arXiv:1801.04381[cs.CV].
18. Keutzer, Kurt (2019-05-22). "Co-Design of DNNs and NN Accelerators" (PDF). IEEE. Retrieved 2019-09-26.
19. Shaw, Albert; Hunter, Daniel; Iandola, Forrest; Sidhu, Sammy (2019). "SqueezeNAS: Fast neural architecture search for faster semantic segmentation". arXiv:1908.01748[cs.CV].
20. Yoshida, Junko (2019-08-25). "Does Your AI Chip Have Its Own DNN?". EE Times. Retrieved 2019-09-26.