NeuraSearchLib: A Modular and Extensible Library for Neural Architecture Search with Configurable Search Spaces

Kaouthar ESSAHELI*, Selsabil ROUBI*, Halima BOUZIDI†, Hamza OUARNOUGHI‡, Smail NIAR‡
* École nationale Supérieure d'Informatique, Algiers, Algeria
†Univ. of California, Irvine, CA, USA
‡ Univ. Polytechnique Hauts-de-France, CNRS, UMR 8201 - LAMIH, F-59313 Valenciennes, France

***Abstract*—While Deep Neural Networks (DNN) have driven technological innovation, designing new architectures remains labor-intensive, requiring human expertise and numerous trial-and-error iterations. Neural Architecture Search (NAS) aims to automate this process for more efficient exploration of DNN architectures. However, current NAS libraries lack flexibility, particularly in DNN search space customization, and lack universal, ready-to-use templates for state-of-the-art NAS search spaces. This paper introduces *NeuraSearchLib*, a modular and extensible NAS library that allows users to create and customize search spaces with high flexibility using simple high-level specifications. *NeuraSearchLib*[1] can reproduce existing NAS search spaces, create new ones, and conduct experiments with various search strategies, training methods, and performance evaluations, enabling researchers to explore new DNN architectures more efficiently.**

***Index Terms*—Deep Learning, Neural Architecture Search, Search Space Design, NAS Library.**

I. Introduction

Deep Neural Networks (DNN) models have garnered significant interest due to their advancements in addressing complex problems, with models like AlexNet [1], VGG [2], and BERT [3] leading breakthroughs in computer vision and natural language processing. Their widespread use in IoT systems has set new standards for performance and efficiency, necessitating innovative and accurate DNN architectures with higher processing efficiency than earlier models. However, manually designing DNN models for specific tasks and systems is labor-intensive and inefficient due to the vast and complex search space of DNN architectures, a challenge exacerbated by hardware constraints such as computation and memory resources.

Neural Architecture Search (NAS) [4] automates the design of DNN architectures, addressing manual approach limitations and abstracting human expertise. A typical NAS framework includes a search space, search strategy, and performance evaluation methods. However, NAS still requires improvements in each component to enhance DNN quality and reduce search and evaluation time, resources, and budget.

Among the primary components of NAS is the search space as it defines and scope of architectures to be explored by the search strategy [9]. These search spaces differ are categorized into: (*i*) Block/Layer-wise search space, which organizes architectures into blocks of operations. (*ii*) Cell-based search space, which consists of a sequence of repeated cells. (*iii*) Hierarchical search space, which represents architectures across multiple levels of architectural patterns. The importance of well-defined search spaces cannot be overstated, as they serve as valuable templates for NAS and exploration. Our goal is to simplify and streamline NAS research by creating a state-of-the-art full NAS library that facilitates the creation of spaces for exploring architectures relevant to the problem and existing knowledge, while also facilitating the integration of new components and ensuring their organization. *Among our challenges is designing a unified search space representation that seamlessly combines various categories into one framework, merging existing space templates. Furthermore, and how to make it easy for users to define precise DNN and search space parameters, aiding in space creation and NAS exploration to uncover relevant DNN architectures*

Over the past years, there has been a significant effort to design comprehensive NAS libraries that integrate the state-of-the-art in NAS. Among the most well-known are NASLib [5], AW-NAS [6], and NNI [7]. We review and compare these libraries in terms of the the decomposition and modularity level in NAS components as well as supported search spaces and the degree of flexibility for each. This comparison is summarized in Table I.

"*On the level of decomposition in the NAS system*. NASLib [5] and NNI [7] coarsely break down the NAS system into basic components, such as search spaces, search strategies, trainers, and performance evaluators, which is a simple decomposition to exploit. However, AW-NAS [6] provides a finer decomposition of NAS into components including objectives (which determine the task), search spaces, controllers, weight managers, trainers, and evaluators. This fine decomposition has advantages but adds complexity during integration.

On the supported NAS search spaces and their flexibility, by "flexibility" we refer to the level of user control over the search space. Low flexibility relates to predefined search spaces without user control, medium flexibility implies basic control over parameters, and high flexibility indicates a high level of control

[1]Our NeuraSearchLib framework will be made open source soon.

TABLE I
COMPARISON BETWEEN EXISTING NAS LIBRARIES AND OUR NEURASEARCHLIB.

Criterion	NASLib [5]	AW-NAS [6]	NNI [7]	NeuraSearchLib (Ours)
NAS System Decomposition level	Coarse-grained	Fine-grained	Coarse-grained	Coarse-grained
Supported search spaces	• Block-wise: None • Cell-based: NasBench-[1,2,3]01 • Hierarchical: Liu's search space [8]	• Block-wise: Mnasnet, Mobilenet • Cell-based: General template • Hierarchical: None	• Block-wise: Hub spaces • Cell-based: NasBench, Hub spaces • Hierarchical: None	• Block-wise: General template • Cell-based: General template • Hierarchical: General template
Level of flexibility (per search space)	• Block-wise: / • Cell-based: **Medium** • Hierarchical: **Low**	• Block-wise: **Medium** • Cell-based: **High** • Hierarchical: /	• Block-wise: **Medium** • Cell-based: **Medium** • Hierarchical: /	• Block-wise: **High** • Cell-based: **High** • Hierarchical: **High**

over operations, architecture topology, etc. NNI [7] offers a range of predefined cell-based and block-wise search spaces, such as NASBench and certain well-known models like NASNet. However, NNI [7] does not provide a general template for a highly customizable search space. To create a new search space, the user must implement it from scratch using abstract classes provided by NNI [7]. NASLib [5], on the other hand, does not provide any examples of block-wise search spaces. However, it offers a general template for cell-based search spaces that users can customize via a configuration file, albeit with limited flexibility. It also provides predefined cell-based search spaces such as NasBench-101/201/301/ASR/NLP. For hierarchical search spaces, NASLib [5] only includes a predefined implementation based on Liu's work [8]. In contrast, AW-NAS [6] offers blockwise search spaces, such as those with MnasNet/MobileNet backbones. However, it does not provide a customizable general format for this type of search space. For cellular search spaces, AW-NAS offers a general format that users can customize with their own parameters. However, hierarchical search spaces are not supported by AW-NAS, and integrating them may require adaptation efforts.

In this work, we address the limitations of existing NAS libraries in terms of customization, space creation with minimal time, effort, and implementation errors, and their integration and manipulation with other NAS components. We propose a modular and extensible NAS library, with the contributions:

- NeuraSearchLib enables the creation of custom search spaces through intuitive high-level specification of universal templates (Blockwise, Cell-based, and Hierarchical).
- Our framework allows users to define most user-defined elements within both deep neural networks and their NAS search space, covering operations, parameters, connections, combination operations between neural operators, and other relevant specifications.

II. THE PROPOSED LIBRARY OVERVIEW

Our library, *NeuraSearchLib*, builds upon the NASLib framework [5] with enhancements at the search space level. We decompose NAS elements into **basic and simple components** without finer segmentation that adds integration complexity. These components are: search spaces, which involve configurable templates of the supported categories of search spaces; search methods, which determine the exploration methods of neural architectures; trainers, which provide the methods used to optimize the weights of DNNs for a specific task; and performance estimations, which provide the methods used to estimate the performance of the model, as demonstrated in Figure 1. NeuraSearchLib is structured into low-level modules representing each component and functionality, **with proper assignment of responsibilities and minimal coupling between modules**. It offers base classes that standardize and factorize methods, allowing easy integration of new components. Additionally, we have a **module that analyzes** the library files at each execution, recognizing new objects without needing to import them into the code. All these factors justify the **modularity and extensibility** of our library.

In addition, we have further refined and customized the search spaces to better suit our requirements. Moreover, we introduce a new component, the *Experiment Manager*, to handle the NAS pipeline by initiating experiments, adjusting their states, and managing the execution of experiment, etc.

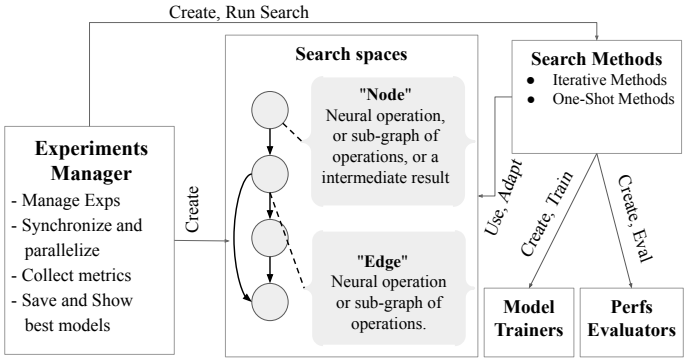

Fig. 1. High-Level Overview of the *NeuraSearchLib* Library: Basic Components and Their Interactions.

A. NeuraSearchLib Search spaces

After thorough analysis, we propose specifications defining the search space and DNN architectures, encompassing neural operations and their parameters, primitive and user-defined operation blocks, as well as network topology parameters such as architecture size, neural operator arrangement, and connections. Users can define fixed and searchable aspects with constraints. Internally, we decouple the search space module from the architecture representation module (which is in the form of a directed acyclic graph for flexibility in creating different forms of DNN architecture) for improved responsibility assignment. Introducing the search spaces:

1) Layer/Block-wise search space: This NAS search space is based on Layer/Block arrangements, where layers are chained linearly with simple operations [10] or non-linearly with complex connections [11], forming a graph. Each layer can receive information from any preceding layer, making it intuitive for developers to build new architectures. This space is particularly useful for testing new architectures and operations through simple composition of layers or blocks.

Searchable parameters typically include the arrangement of operations, chain size, and operation parameters [9]. We consider this space as a sequence of nodes, where each node represents an operation. Users can specify supported operations and their parameters or indicate that a node is similar to another. Connections between nodes represent information transfer, with default sequential connections that can be customized. Nodes can receive information from any preceding node, allowing users to specify information combination operations. Users can define a range of parameters either within each operation or in a shared "others-config" section. Future adaptations for more complex connections can be made with minimal modifications. Below is the configuration file:

```
primitive_operations: #Neural operations.
  - Conv2d-kernel_size: 5x5 #PyTorch primitive-op
  - SeparableConv2d-kernel_size: 3x3&out_channels:32
     #a user-defined operation block with its params
chain_size: [min, max]
other_configs: # Specifies other parameters
  out_channels: # possible values
  other_parameters: ...
node_operations: #operations supported in each node.
  - node_id:
    allowed_operations: [ op1, op2] # OR
    similar_to: # node_id
    combine_op: # the type of comb-operation.
edge_operations: {from: source_id, to: receiver_id}
    # connections between nodes.
```

Listing 1. Layer/Blockwise search space Configuration File Example

2) Cell-based space: With advancements in NAS and the discovery of interesting patterns, the cellular search space emerged to constrain the search. In this space, architectures consist of repeated patterns (cells) [9], with both cell and arrangement predefined. This format helps users narrow down the search and facilitates the construction of super-networks by limiting the search to cell operations.

Inspired by AW-NAS [6], our library allows users to specify cell groups (name and ID) and their internal topology, including the number of nodes, input connections, and initial nodes from the previous cell. Users can also define supported operations and parameters for each cell group or share them across all groups. This is useful for predefining specifications for various cell types (normal cell, reduction cell, etc.).

In each cell, intermediate nodes use the sum operation to combine information, and the final node uses the concat operation, with future customization options available. Then, the arrangement of these cells is defined by the user (they define the order and disposition of these defined cell groups). Find below the configuration file:

```
num_cell_groups: #Specifies id and name of the cell
    group [0: normal, 1: reduction, 2: new_group  #
    Specifies new group]
cells_layout: [0,0,1,2] # Specifies the layout of
    cell groups.
cell_groups_details:#for each cell group.or ll be
    shared
  num_nodes_per_cell: 0: 4 ,  1: 3 , 2: 3
  num_init_nodes: ....   num_init_inputs:
  primitive_operations: []
```

Listing 2. Cell-based search space Configuration File Example

3) Hierarchical space: Hierarchical search spaces aim to expand the scope of DNN architectures with more complex patterns. In this space, patterns are organized into complex structures composed of sub-patterns [9], useful for tasks like segmentation or data quality improvement.

Our library represents this space as a large graph of patterns, where the nodes represent information (or results of previous operations) and the edges represent operations that are lower-level patterns. Each pattern is a graph with edges as sub-patterns (recursion). Users can specify the number of levels in the graph, the internal topology of each pattern level, and constraints such as specific operations on the edges of the main graph. Each pattern is a dense, directed acyclic graph with nodes linked to all preceding nodes. Each link is an operation: normal, identity, or Zero (no link). Intermediate nodes use the sum operation to merge information, and the final node uses the concat operation. Below is the configuration file:

```
primitive_operations: ...
hierarchical_params: # Hierarchical search space
    parameters
  num_levels: 3  # Number of levels
  details_levels:
    - num_level: 1 # pattern level (level 0
    indicates primitive or blocks of operations)
      num_nodes: 4  # Number of nodes in this
    pattern
    - num_level: 2 # Cell level
      num_nodes: 5  # Number of cells
    - num_level: 3 # External graph level
      num_nodes: 8  # Number of nodes
      edge_operations: {from: 0, to: 1,
    allowed_operations: ...}
    ...
other_configs:
```

Listing 3. Hierarchical search space Configuration File Example

B. NeuraSearchLib Search Methods

Our library support iterative and One-Shot methods through a unified interface, with an adaptation function within the search method, which takes the search space and adjusts it accordingly. NeuraSearchLib supports various strategies like random search, Bayesian optimization [11], and regularized evolution [12], Additionally, users can easily integrate others.

C. NeuraSearchLib Training and Performance Estimation Methods

Our library offers basic training methods, direct evaluation, and a Bayesian surrogate estimator [11]. Users can easily incorporate additional surrogate models and predictors for performance estimation and others. The neural network

modules generated by our library allow for the extraction of key attributes and architectural parameters, enhancing their usability for encoding and other purposes.

III. Evaluation and Case Studies

We showcase that our library enables running a complete NAS experiment with search spaces used in practical use-cases. We further demonstrate that our general template for search spaces can effectively reproduce SOTA DNN models.

A. Random search on a layer-wise space

We reproduce the MobileNetV2-Space search space from [13]. This space consists of MBConv modules with a fixed macro architecture. The search components within each MBConv module are the kernel size and the expansion rates/channels. We run the experiment with a random search method, using a simple trainer and evaluator. Below is the configuration file, which fixes a block operation in each node and provides a range of relevant searchable parameters:

```
search_space: LayerSearchSpace:
  config:
    chain_size: [19, 19]
    node_operations:
      - node_id: 1
        allowed_operations: [Conv2d-kernel_size:3x3&
    out_channels:32]
      - node_id: 3  # ...
        allowed_operations: [MBConv-out_channels:24]
      - node_id: 4
        similar_to: 3 #this replace repeat block
    function
      - node_id: max  # Last node with allowed
    operation
        allowed_operations: [AvgPool2d-kernel_size:1
    x1]
        other_configs: {kernel_size: [3, 5],
    expansion_ratio: [1,2, 3, 4 ,6]}
search_method:
  method: RandomSearch
  ...
```

Listing 4. Sample Config Section for Reproducing MobileNetV2-Space

Since the macroarchitecture of MobileNetV2 is fixed, variations occur at the parameter level, therefore, the MobileNetV2 model belongs to this space. We can also vary or set an operation in a searchable node with allowed operations to search for or introduce skip connections to test modified architectures based on the MobileNetV2 space. After launching this experiment, we generated MobileNetV2 architectures and other relevant architectures sharing the same macroarchitecture but with different parameter values. Conducted on CIFAR10 with minimal data and search time, the experiment yielded models with over 87% accuracy.

B. Regularized evolution on a cell-based search space

We implement a search space inspired by and compatible with the ENAS [14] and DARTS search spaces, taken from the aw-nas [6] library. Users can choose the arrangement and types of cells (normal and reduction), and the internal topology (number of nodes and initial edges in each cell group). Each cell is considered a graph. We use a regularized evolution search method. Below is the configuration file:

```
search_space:
    pattern: CellSearchSpace
    config:
        num_cells: 8
        num_cell_groups: {0: normal, 1: reduction}
        cells_layout: [ 0,0, 1,0, 0,1, 0,0 ]
        cell_groups_details:
            num_nodes_per_cell: 4
            num_init_nodes: 2
            num_init_inputs: 2
            primitive_operations: [Zero,Identity,
    MaxPool2d-kernel_size:3x3,SeparableConv2d-
    kernel_size:5x5,DilatedConv2d]
            other_configs: {out_channels: [32, 64]}
search_method:
  method: RegularizationEvaluation
```

Listing 5. Sample Config Section for Reproducing ENAS Search Space

We successfully reproduced the targeted space (cell groups and their arrangements) with minimal time and implementation effort. We achieved top-5 model accuracy between 71% and 89% on CIFAR-10 for this configuration.

C. Random search on a hierarchical search space

We reproduce the hierarchical search space as defined in [8] and inspired by NAS-Lib [5]. We define three levels: **Level 0:** Primitive operations. **Level 1:** Motifs (dense subgraphs with 4 nodes). **Level 2:** Cells (dense subgraphs with 5 nodes, with edges as lower-level operations). **Level 3:** External graph (a continuous graph of 14 nodes, with edges as lower-level graph operations). Some edges of this graph are predefined (starting stem operation block and certain edges with Sep-Conv), while others are searchable. We conduct a search experiment using the random search method. Below is the configuration file:

```
search_space: HierarchicalSearchSpace:
  primitive_operations: [SepConvDARTS-stride:1x1,
    Identity,MaxPool2d-kernel_size:2x2, #..  ]
  hierarchical_params:
    num_levels: 3
    details_levels:
      - {num_level: 1, num_nodes: 4}  # Motif
      - {num_level: 2, num_nodes: 5}  # Cell
      - {num_level: 3, num_nodes: 14,
    edge_operations: [  # External Graph
          {from: 0, to: 1, allowed_operations: [
    StemNASLib]},
          {from: 2, to: 3, allowed_operations: [
    SepConvDARTS-out_channels:16&kernel_size:3&
    stride:1&padding:1]},
          # ...
        ]}
  other_configs: {out_channels: [16, 32, 64], stride
    : [1, 2], kernel_size: [3, 5], expansion_ratio:
    [1, 6]}
search_method: RandomSearch
```

Listing 6. Sample Config Section for Reproducing Hierarchical Search Space

The space respects the targeted area with specific operations, edges, levels, and motif topology. It is larger than the space defined in the article because it considers motifs from all lower levels, not just the previous one. This approach generates diverse architectures, including those within the targeted space, achieving top-5 model accuracy between 80% and 87% on CIFAR-10.

IV. Conclusion

We introduced NeuraSearchLib, a modular library for Neural Architecture Search (NAS) that allows easy creation of custom search spaces with universal templates. It simplifies the search process and helps discover relevant DNN architectures. Future works include optimizing performance, supporting more architectures and operations, addressing complex deep learning tasks, and reducing dependency on frameworks like PyTorch.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 1

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

[4] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. 1

[5] Michael Ruchte, Arber Zela, Julien Niklas Siems, Josif Grabocka, and Frank Hutter. Naslib: A modular and flexible neural architecture search library. 2020. 1, 2, 4

[6] Xuefei Ning, Changcheng Tang, Wenshuo Li, Songyi Yang, Tianchen Zhao, Niansong Zhang, Tianyi Lu, Shuang Liang, Huazhong Yang, and Yu Wang. aw_nas: A modularized and extensible nas framework. *arXiv preprint arXiv:2012.10388*, 2020. 1, 2, 3, 4

[7] nni. https://github.com/microsoft/nni. 1, 2

[8] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017. 2, 4

[9] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023. 1, 3

[10] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 3

[11] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31, 2018. 3

[12] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 3

[13] Krishna Teja Chitty-Venkata and Arun K Somani. Neural architecture search survey: A hardware perspective. *ACM Computing Surveys*, 55(4):1–36, 2022. 4

[14] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018. 4