Visual Analysis of Neural Architecture Spaces for Summarizing Design Principles

Jun Yuan, Mengchen Liu, Fengyuan Tian, and Shixia Liu

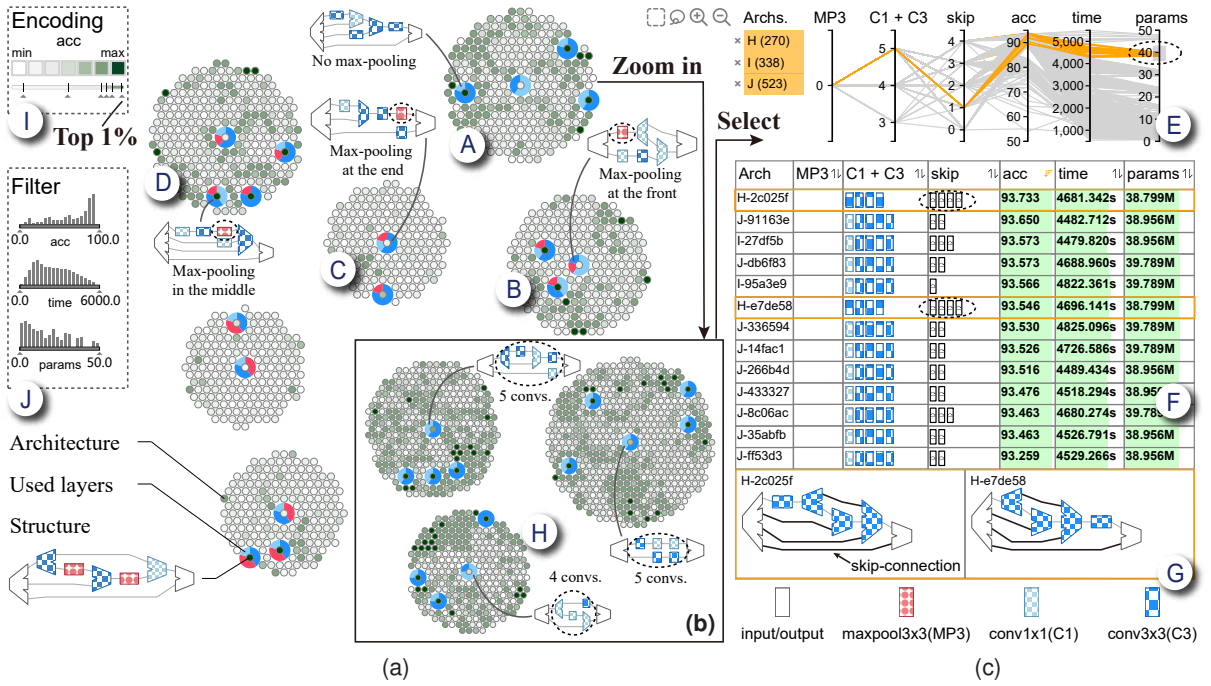

Fig. 1: ArchExplorer: (a) the architecture visualization to show the architecture clusters; (b) three selected sub-clusters after zooming into cluster A; (c) a detailed comparison of the selected architectures.

Abstract— Recent advances in artificial intelligence largely benefit from better neural network architectures. These architectures are a product of a costly process of trial-and-error. To ease this process, we develop ArchExplorer, a visual analysis method for understanding a neural architecture space and summarizing design principles. The key idea behind our method is to make the architecture space explainable by exploiting structural distances between architectures. We formulate the pairwise distance calculation as solving an all-pairs shortest path problem. To improve efficiency, we decompose this problem into a set of single-source shortest path problems. The time complexity is reduced from $O(kn^2N)$ to $O(knN)$. Architectures are hierarchically clustered according to the distances between them. A circle-packing-based architecture visualization has been developed to convey both the global relationships between clusters and local neighborhoods of the architectures in each cluster. Two case studies and a post-analysis are presented to demonstrate the effectiveness of ArchExplorer in summarizing design principles and selecting better-performing architectures.

Index Terms— Machine learning, visual analytics, neural architecture search, design principle, knowledge discovery.

1 Introduction

Recent progress in artificial intelligence largely benefits from better neural network architecture design [19, 33]. The successful design of these architectures relies on costly trial-and-error processes. With the development of large GPU clusters, neural architecture search (NAS) [17] has been proposed to parallel the architecture design process. It automatically selects well-performing architectures in neural architecture spaces by training and evaluating a large number of architecture candidates. Since the NAS method aims to find well-performing architectures for given datasets, it depends on the evaluation of the specific datasets. Accordingly, the generalization ability of the searched architectures may be limited. Design principles, which describe how specific structure components, such as layers or their combinations, influence the performance of architectures, have been shown to be useful in designing more explainable architectures with better generalization ability [58]. Moreover, they can be used to reduce the search space and computation cost of the NAS method. A recent study indicates that a comprehensive analysis of architecture spaces facilitates the summarization of such design principles [49]. It is therefore of theoretical and practical significance to analyze these spaces for advancing our understanding of the influence of the structure on model performance.

There are two technical challenges in analyzing an architecture space. First, the number of architectures brings the scalability issue. Previous research has demonstrated that understanding the structural distances between architectures enables users to derive general design principles for architecture design [58]. However, the space usually contains tens of thousands to millions of architectures [35], which leads

- *J. Yuan, F. Tian and S. Liu are with BNRist, Tsinghua University. E-mail: {yuanj19, tianfy21}@mails.tsinghua.edu.cn, shixia@tsinghua.edu.cn. Shixia Liu is the corresponding author.*
- *M. Liu is with Microsoft. E-mail: mengcliu@microsoft.com*

Manuscript received 31 March 2022; revised 1 July 2022; accepted 8 August 2022. Date of publication 3 October 2022; date of current version 2 December 2022. This article has supplementary downloadable material available at https://doi.org/10.1109/TVCG.2022.3209404, provided by the authors.
Digital Object Identifier no. 10.1109/TVCG.2022.3209404

to at least millions of distance calculation. Thus, how to efficiently calculate so many distances is still an open question. Second, it is non-trivial to identify the architectures of interest and analyze them in context for summarizing design principles. Given a large number of architectures, a scatterplot is commonly used to show the performance (*e.g.*, accuracy or speed) versus a numerical property associated with the architectures (*e.g.*, the number of parameters or floating-point operations). Although it can provide a performance overview of the architectures, it fails to reflect their structural distances. This hinders the understanding of the structural connections between architectures, and thus brings difficulty in summarizing design principles. It is therefore technically demanding to provide an interactive exploration environment where structurally similar architectures are placed together, and smooth exploration is supported to probe the architecture space from global overview to individual architectures.

In this work, we propose a visual analysis method, ArchExplorer, to facilitate the interactive analysis of an architecture space. Most neural network architectures are composed of a few sub-architectures repeated multiple times [31, 61], each of which is a combination of multiple layers (*e.g.*, convolution layers and pooling layers) and their connections [15]. Zoph *et al.* have demonstrated that stacking well-performing sub-architectures can construct state-of-the-art architectures [77]. Thus, we focus on analyzing the repetitive sub-architectures. Without loss of generality, we refer to them as architectures in the discussion below. We represent each architecture as a directed acyclic graph (DAG) and adopt the widely-used graph edit distance to measure the structural distances between them. We formulate the calculation of all pairwise structural distances as an all-pairs shortest path problem. To efficiently calculate millions of pairwise distances or even more, we decompose this problem into a set of single-source shortest path problems. They can be solved by an accelerated Dijkstra algorithm. Our distance calculation algorithm reduces the time complexity from $O(kn^2N)$ to $O(knN)$. Using the calculated distances, we build an architecture cluster hierarchy to enable an efficient exploration of such a large space. An architecture visualization is then designed for better understanding the architecture space. To help efficiently identify the architectures of interest, a force-directed layout is employed for preserving the global relationships between clusters at each hierarchy level. To facilitate the analysis of the architectures in context, a circle packing layout is developed for illustrating the local neighborhood of the architectures in each cluster (Fig. 1(a)). Coupled with a set of interactions, such as zooming and comparison, this visualization enables users to summarize design principles. We conduct two case studies on two NAS benchmark datasets to demonstrate the capability of ArchExplorer in deriving design principles. A post-analysis with a state-of-the-art method, LaNAS [63], shows that the derived principles can reduce the computation cost for searching the better-performing architectures. A demo of the prototype is available at: `http://archexplorer.thuvis.org`.

The key technical contributions of this work are:

- **The formulation** of the pairwise distance calculation as solving an all-pairs shortest path problem.
- **An architecture visualization** that preserves both the global relationships between architecture clusters and the local neighborhoods of architectures in each cluster to facilitate the identification and comparison of the architectures of interest.
- **A visual analysis tool** to understand an architecture space and summarize general design principles through the analysis of a large number of neural network architectures.

2 Related Work

We briefly review two categories of related work: explaining machine learning models and explaining automated machine learning methods.

2.1 Explaining Machine Learning Models

Many visual analysis methods for explainable deep learning have been developed to facilitate the analysis of machine learning models [21, 53, 74]. According to the analysis goal, they can be categorized into two classes: diagnosis-oriented analysis and comparative analysis [21].

Diagnosis-oriented analysis aims to explain model behaviors and diagnose models with unsatisfactory performance. Most existing efforts focus on revealing the working mechanisms of different models, such as multilayer perceptrons [50], ensemble models [38, 72, 76], convolutional neural networks [28, 37, 67], deep generative models [36], recurrent neural networks [41, 57], and Transformers [14, 24, 34]. Despite their effectiveness in analyzing a single model, they do not support model comparison, which is essential for selecting better-performing models from a set of candidates.

To fill this blank, **comparative analysis** methods are developed to explain the similarities and differences between models, therefore providing guidance for model selection. Such visual analysis methods have been developed for diverse tasks, such as classification [39, 43, 51] and question answer verification [14]. For example, Murugesan *et al.* [43] proposed DeepCompare to compare the error patterns between models. This is achieved by analyzing their differences in the neuron weights and activations. These methods facilitate model comparison, but they are less capable of revealing the full picture of an architecture space and identifying the architectures of interest. Thus, they are not efficient in selecting a well-performing architecture in a large space. In comparison, by preserving the pairwise distances between architectures, ArchExplorer provides an overview of the architecture space and also enables the analysis of individual architectures. Thus, it empowers users to summarize design principles for designing better-performing architectures.

2.2 Explaining Automated Machine Learning Methods

Automated machine learning aims to automate the tedious and iterative tasks in building a machine learning model, including data preprocessing, feature selection, architecture design, and hyper-parameter tuning [20]. Accordingly, researchers have developed several visual analysis methods to analyze the automated generated results of these tasks [6, 7, 9, 13, 47, 64, 65] or their combinations [46, 62, 68].

Our work falls in the category of architecture-design-oriented works [6, 7, 13]. The methods seek to discover better architectures. Among these research attempts, the most relevant one is REMAP [7], which is one of the pioneering works in visually analyzing neural architecture spaces. It provides an effective iterative process for designing sequential neural network architectures. Starting from a set of random architectures, users can design new architectures by modifying the structures of the selected ones. These modifications can be recommended by the system or specified by users. REMAP can help users efficiently build a well-performing architecture. It pays less attention to summarizing design principles from a large neural architecture space. In addition, the employed MDS projection may fail to place structurally similar architectures together [59]. In contrast, ArchExplorer focuses on summarizing design principles from an architecture space. To this end, we first develop an efficient distance calculation algorithm and build an architecture cluster hierarchy. Based on them, an architecture visualization is designed. It places structurally similar architectures together and allows users to analyze the architectures in the context of similar architectures.

3 Design Requirements

ArchExplorer is developed in collaboration with four experts in computer vision (E_1 – E_4) whose research interests include image classification and object detection. In one project, they utilized the NAS method to develop an efficient image classification model for an online content tagging service. To improve the generalization of the searched architecture, the experts employed several visualizations to help them derive design principles. For example, they used a scatterplot to gain an overview of the searched architectures. The X-axis and Y-axis represent inference time and accuracy, respectively. Such visualization gives them an overall idea of the searched architectures. However, there lacks a comprehensive understanding of the structural relationships between architectures. Without such an understanding, it is difficult to link the structural differences of architectures with their performance differences. This brings difficulty in discovering design principles, such as whether a structure component in an architecture is beneficial to the performance. Thus, it is desired to develop a visual analysis tool to help summarize such design principles from a large architecture space.

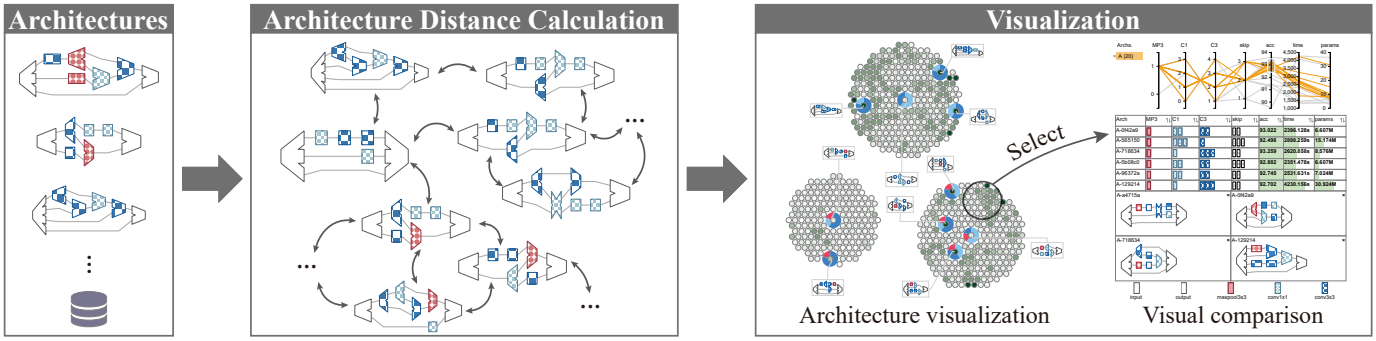

Fig. 2: ArchExplorer overview: given the architectures in an architecture space, the **architecture distance calculation** module calculates the distances between them; the **visualization** module provides an overview of the architecture space and helps users compare architectures.

In the past two years, we held biweekly free-form discussions with the experts to probe the requirements and develop the tool. Based on the discussions, we summarized the following design requirements:

R1. Grouping architectures based on their structural distances. All experts expressed the need to get a clear understanding of an architecture space with hundreds of thousands of architectures or even more. To analyze such a large space, the experts usually cluster the architectures into several groups. The widely used practice is to group architectures by the numerical properties such as inference time or accuracy [49, 73]. Such a grouping strategy can help find well-performing architectures. However, the experts commented that it usually hindered the discovery of design principles due to various structures in each architecture group. Recent research has shown that the structure of a neural network architecture is one important factor that influences its performance [30]. Thus, the experts required to model the structural distance between architectures and build the clusters accordingly.

R2. Identifying the architectures of interest and analyzing them in context. Currently, to analyze an architecture space and select the ones of interest from it, the experts plotted all the architectures in a scatterplot, where each point encodes an architecture. With this scatterplot, the experts can analyze these architectures by different numerical properties (*e.g.*, accuracy, inference time, the number of parameters, etc.) represented by the axes. Although such visualization can provide a numerical overview of the architectures, it fails to reflect their structural distances and hinders the discovery of similar architectures associated with the ones of interest. Thus, to discover design principles from a large space, it is demanding to provide a more informative overview that can be served as an entry point for the analysis. Once identifying the architectures of interest, it is desirable to examine the architectures in context. As explained by E_2, "Comparing an architecture of interest with its neighbors is similar to conducting an ablation study that is useful to understand how different structure components of the architecture contribute to the performance and whether a specific structure component is beneficial to the performance or not."

R3. Comparing architectures in multiple aspects. The experts commented that architecture comparison was the key to deriving design principles. This is also consistent with the findings of recent research on deep model comparison [39, 43]. The experts indicated that a design principle usually came from comparing two architecture groups with different accuracy. E_3 said, "Among computer vision researchers, there are different opinions whether layer normalization should be positioned before or after the attention block in Transformer models. To verify this, I constructed pairs of transformers with different widths and depths. In each pair, layer normalization is put before and after the attention block, respectively. By comparing their accuracy, I find that putting layer normalization before attention is overall beneficial for the classification task." In addition, the experts emphasized that making a decision on architecture design often involved multiple criteria, and they needed to compare the architectures of interest in multiple aspects. For example, besides accuracy, E_1 also wanted to compare other measures, such as inference time and the number of floating-point-operations (FLOPs). As E_1 further explained, the number of FLOPs is positively related to the performance and the energy cost of GPUs [54]. Thus, he needed to compare the architectures with different numbers of FLOPs and select one that can well balance the performance and the number of FLOPs.

4 Design of ArchExplorer

Driven by the design requirements, we develop ArchExplorer to support the interactive analysis of an architecture space and summarize design principles. It contains two main modules: architecture distance calculation and visualization (Fig. 2). The former models the distances between architectures (**R1**). The latter first builds an architecture cluster hierarchy based on the distances, and then allows users to quickly identify the architectures of interest and analyze them in context (**R2**). It also facilitates the comparison of the architectures to understand which structure components are beneficial to the performance (**R3**).

4.1 Calculation of Architecture Distance

Since NAS algorithms aim at searching for a well-performing architecture, existing distance-based NAS algorithms only calculate the distances between each of the newly selected architectures and the previously evaluated ones [25, 30]. However, ArchExplorer aims at grouping structurally similar architectures together (**R1**), which requires the pairwise distances between all architectures.

Problem formulation. The appropriate structure-based architecture representation is the key to modeling the distances between architectures. There are two common schemes for representing the structure of an architecture: path-based and DAG-based [69]. The path-based scheme represents an architecture as a set of paths, where each path is a sequence of layers. The DAG-based scheme represents an architecture as a directed acyclic graph (DAG), where nodes and edges represent its layers and the connections between layers. A previous study has shown that the DAG-based scheme explicitly utilizes the topological information and better reflects the performance of a neural network architecture [45]. Thus, in ArchExplorer, we adopt the DAG-based scheme to represent architectures. With this representation, we adopt the widely-used graph edit distance (d) to measure the structural distances between architectures [25]. The graph edit distance between two architectures is defined as the minimum cost of all possible edit paths that transform one architecture into another [52].

The A* algorithm is commonly used for calculating graph edit distances [2]. Given two architectures, this algorithm finds the minimum-cost edit path by iteratively exploring the architectures with one edit operation difference. Assume that there are N architectures in the whole architecture space, and each architecture can be transformed into k different architectures with one edit operation. The time complexity for calculating the graph edit distance between two architectures is $O(kN)$. In practice, the whole architecture space can probably contain billions of architectures or even more [56, 60], which raises difficulty in the analysis. Typically, n architectures ($n < N$) are sampled from the whole space to ease the analysis. Given n sampled architectures, directly applying the A* algorithm for calculating the pairwise distances results in the time complexity of $O(kn^2N)$. This is still very time-consuming, especially for hundreds of thousands of sampled architectures.

We observe that the minimum-cost edit paths between different architecture pairs can overlap, such as the path $x-v$ in the paths $x-v-z$

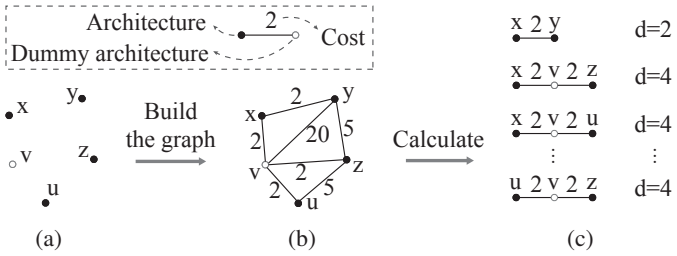

Fig. 3: Architecture distance calculation: (a) an example architecture space; (b) building the graph by connecting architectures with only one edit operation difference; (c) calculating all the pairwise distances.

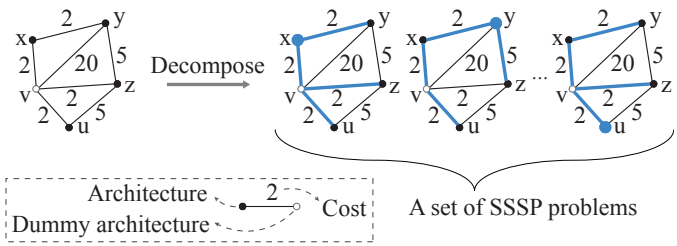

Fig. 4: Decomposing the APSP problem into a set of SSSP problems. The blue dots represent the source architecture, and the blue lines are the visited paths from the source architecture in the Dijkstra algorithm.

and $x-v-u$ in Fig. 3(c). This indicates that the minimum-cost edit paths have the optimal substructure property, which enables the reuse of previously found minimum-cost edit paths. Motivated by such an observation, we propose to model the previously found minimum-cost edit paths and the associated nodes as a graph. Accordingly, we build an architecture graph by connecting the architectures with only one edit operation difference to represent their one-hop neighbor relationships (Fig. 3(b)). For n sampled architectures, we add $N-n$ dummy architectures to guarantee the existence of the minimum-cost edit path between any two architectures. The dummy architectures are those in the architecture space but not in the sampled architectures. Fig. 3(b) shows an example architecture graph. In this graph, each solid dot represents an architecture, and each hollow dot represents a dummy one. The weight on each edge encodes the cost of the edit operation. In our implementation, we utilize the cost matrix given by Nguyen *et al.* [44] to define the substitution cost (w_e) between different layers. The deletion (addition) cost is defined as the substitution cost between a given layer (the null layer) and the null layer (a given layer), which is set as $5\times\max_e w_e$. Thus, calculating the pairwise edit distances in this architecture graph can be formulated as an all-pairs shortest path (APSP) problem (Fig. 3(c)).

Algorithm. Since the number of one-hop neighbors of an architecture (k) is much smaller than the number of architectures in the whole architecture space (N), the architecture graph is sparse. Due to such sparsity, an efficient method for finding the minimum-cost edit paths is to decompose the APSP problem into a set of single-source shortest path (SSSP) problems for each source architecture (Fig. 4). In each SSSP problem, we employ an accelerated Dijkstra algorithm to find the minimum-cost edit paths between the source architecture and the others. Dijkstra algorithm utilizes a greedy strategy to iteratively find the shortest paths between nodes based on the edge costs [12]. The most time-consuming step at each iteration is the selection of a node that has the minimum cost to the source ($O(\log N)$). Since the edit operations are finite (insertion, deletion, and substitution of a node/edge), the associated costs are also finite. We use this property to improve efficiency. In particular, we maintain sets of architectures, each of which consists of the architectures with the same cost to the source. These sets are organized as a sorted list based on their costs. With this sorted list, the algorithm can select an architecture with the minimum cost in constant time ($O(1)$) [42].

Complexity analysis. As each architecture has k one-hop neighbors, our algorithm takes $O(kN)$ time to build the architecture graph with kN edges. It runs n times of the accelerated Dijkstra algorithms, and each has an $O(kN)$ time complexity. Accordingly, the total time complexity of our algorithm is $O(kN)+n\times O(kN)=O(knN)$. It is faster than the A* algorithm ($O(kn^2N)$) with an acceleration ratio of n. For example, in the case studies, our algorithm can achieve an acceleration ratio of 423,624 and 15,625 on the NAS-Bench-101 [73] and NAS-Bench-201 [16] architecture spaces, respectively. The developed algorithm works well when N is no more than several millions. When N reaches billions or even larger, our algorithm fails because both the time complexity and space complexity are proportional to N. For example, we are unable to build an architecture graph for NAS-Bench-301 [56] as it contains over 10^{18} architectures. A common solution for handling such large spaces is to directly calculate the pairwise distances between the sampled architectures. The pairwise distance is the minimum matching cost among all possible matchings between the layers of the two associated architectures. The time complexity for calculating a distance is $O(L!)$, where L is the total number of layers in the two architectures. When $L>9$, $L!$ grows more than a million. It is intractable to directly compute all these distances ($O(n^2L!)$). To tackle this issue, we represent all possible matchings by a weighted bipartite graph and utilize an approximation algorithm to accelerate the calculation [52]. A sub-optimal matching can be found in $O(L^3)$.

4.2 Architecture Visualization

The architecture visualization is designed to facilitate the analysis of the architecture space from a global overview to individual architectures. Based on the calculated distances, we build an architecture cluster hierarchy in a top-down manner by iteratively applying K-medoids [48]. This algorithm is widely used to cluster samples with distance measures due to its simplicity, fast convergence, and robustness to noise [3]. The number of clusters is determined by a grid search, selecting the one with the minimum average distance (to the cluster center). Then we employ a cluster-aware sampling strategy [75] to sample architectures from each level of the architecture cluster hierarchy for display. This sampling strategy aims to maintain the relative sizes of clusters. To preserve smaller clusters, it also guarantees sampling a minimum number from each cluster. In our implementation, we set this number as 10.

4.2.1 Architecture Space as Circle Packing

Visual design. As shown in Fig. 5(a), an architecture is represented by a circle. A sequential color scheme is utilized to encode accuracy. The darker green the circle, the higher the accuracy. By default, the darkest green circles highlighted the well-performing architectures with top 1% accuracy (Fig. 1I). Architectures in the same cluster are densely packed together. We employ circle packing to avoid overplotting in scatterplots and reduce the learning curve of users because it shares the same visual encoding with the widely-used scatterplots. To connect the performance to the structures of individual architectures, we select representative architectures in each cluster and visualize them with summary glyphs (Fig. 5(b)) and structure glyphs (Fig. 5(c)), which provide different levels of structural information. The summary glyph uses a doughnut chart around the circle to show the ratio of different layers used in the

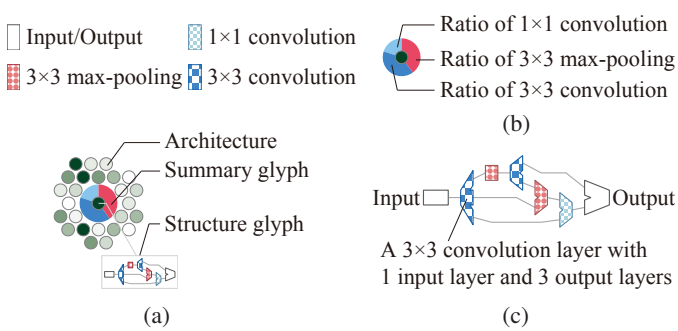

Fig. 5: Visual design of the architecture space: (a) an architecture cluster; (b) a summary glyph; (c) a structure glyph.

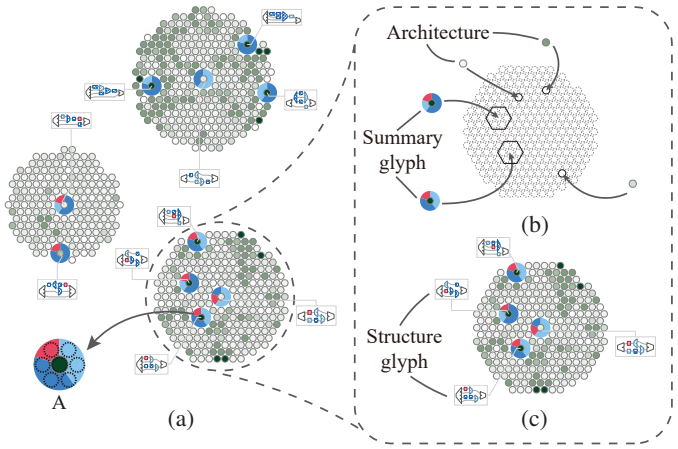

Fig. 6: The architecture layout: (a) global cluster layout; (b) local architecture and summary glyph packing; (c) structure glyph placement.

representative architecture. The color and length of each arc encode the layer type and the ratio of the layer used, respectively. The structure glyph employs a simplified architecture representation, Net2Vis [5], to illustrate the structure of the representative architecture. It provides general information about how individual layers are connected.

Layout. The architecture visualization aims to illustrate the similarity relationships between architectures (**R2**), including the global cluster relationships and local neighborhoods of architectures. To this end, a force-directed layout is utilized to place the architecture clusters based on their distances to each other (Fig. 6(a)), and a circle packing layout algorithm is developed to preserve the local neighborhoods of the architectures within each cluster (Fig. 6(b)). We then enhance this visualization by appending structure glyphs to the representative architectures (Fig. 6(c)), which helps to reveal the structural characteristics of each cluster. The widely used Kamada-Kawai layout algorithm [29] is employed to place the clusters, where we use the distances between the cluster centers to represent the distances between clusters. Here, we focus on introducing the circle packing and glyph placement.

The goal of the circle packing is to preserve the local neighborhoods of architectures. However, the commonly used circle packing algorithm, the front-chain algorithm [66], pays little attention to preserving local neighborhoods. As the densest packing of identical circles is equivalent to the regular hexagonal packing on the plane [8], we transform the circle packing into a hexagonal grid layout problem. Here, the densest packing is an arrangement that maximizes the number of packed circles in a given layout area. By considering the similarity relationships between architectures globally, the hexagonal grid layout places similar architectures adjacently. The optimal layout is generated by minimizing the sum of distances between the adjacent architectures in the grid. Assume that cluster i contains N_i architectures $\{a_j\}_{j=1}^{N_i}$, and we generate a grid containing N_i grid points $\{x_j\}_{j=1}^{N_i}$. Let Π_{N_i} be the set of all possible permutations of $\{1,2,\cdots,N_i\}$. Then a layout can be represented by a permutation $\pi \in \Pi_{N_i}$ where a_j is placed on $x_{\pi(j)}$. Accordingly, the hexagonal grid layout problem is formulated as:

$$\underset{\pi\in\Pi_{N_i}}{\text{minimize}} \quad \sum_{j=1}^{N_i} \sum_{x_{\pi(l)}\in\Gamma(x_{\pi(j)})} d(a_j,a_l). \tag{1}$$

Here, $\Gamma(x_{\pi(j)})$ is the set of adjacent grid points of $x_{\pi(j)}$ and $d(a_j,a_l)$ is the distance between architectures a_j and a_l that are assigned to adjacent grid points $x_{\pi(j)}$ and $x_{\pi(l)}$, respectively. Since this is an NP-hard quadratic assignment problem [1], we propose a greedy algorithm to effectively generate an approximate layout result. Specifically, for each grid X_i, we first sort the grid points by their distances to the center of X_i. Then each architecture is iteratively assigned to a grid point that leads to the maximal decrease in the sum of distances to get an initial feasible result. Finally, we tune the result by swapping the architectures assigned to two grid points that can further decrease the sum of distances until no more such swaps can be performed.

To better understand the structural characteristics of the clusters, 2-5 representative architectures are selected from each cluster. To well represent the cluster and also prioritize the architectures with higher accuracy for analysis, the representative architectures are either with 1) the top-1 similarity to other architectures; or 2) the top-10 accuracy balancing accuracy and similarity. Each representative architecture is associated with a summary glyph and a structure glyph for providing more structure-level information. To uniformly pack the summary glyphs and the circles together, one summary glyph takes up seven grid points, including the grid point associated with the corresponding architecture and all its adjacent grid points (Fig. 6A). In this way, packing circles and summary glyphs together can still be regarded as assigning them to the grid points and solved with the proposed layout algorithm. We then utilize the label layout algorithm [40] to place the structure glyphs near their corresponding summary glyphs. Following the Gestalt law of connectedness [32], we link the structure glyph to the corresponding summary glyph to strengthen their visual connection.

4.2.2 Interactive Exploration

To facilitate the exploratory analysis of the architecture space and help users identify the architectures of interest for detailed analysis, a set of interactions is provided.

Filtering. Following the design in [10], we provide a set of scented widgets (Fig. 1J) to filter out irrelevant architectures. They guide the architecture filtering by displaying their attribute distributions (*e.g.*, the number of FLOPs and parameters).

Selection. Two modes are provided to select the architectures of interest. Users can select the ones in a specific cluster by the cluster mode ( ) or from an arbitrary region by the lasso mode ( ).

Navigating through different levels of detail. With the architecture cluster hierarchy, users can click to zoom into a specific architecture cluster and examine the fine-grained sub-clusters or to zoom out to the previous navigation level. To keep users' mental map, the sampled architectures at the current level are preserved when zooming into a specific cluster. We also maintain the layout stability by keeping the relative position unchanged across different navigation levels.

Comparison. A comparative visualization (Fig. 1(c)) is designed to enable a detailed comparison of the selected architectures in three aspects (**R3**). First, we use a parallel coordinates plot (PCP) (Fig. 1E) to compare the attribute distributions between **groups** of architectures due to its effectiveness in comparing different numerical attributes [26]. In the PCP, each polyline represents an architecture, and each axis represents an attribute. Second, a table is utilized to compare **individual** architectures (Fig. 1F), which is enhanced by encoding each attribute with both a numerical value and a bar Value in one cell. Third, we enable a side-by-side comparison of architecture structures (Fig. 1G), which is critical in summarizing design principles. They are visualized with the same design as the structure glyphs (Fig. 5(c)).

5 Evaluation

To demonstrate the effectiveness of ArchExplorer in summarizing design principles, we conducted two case studies with the experts. To further validate the usefulness of the summarized design principles, we integrated them into a state-of-the-art NAS method. Experimental results showed that the principle-based NAS method reduced the computation cost by around 50% and achieved at least the same performance as the NAS method. The accuracy of each architecture was evaluated on the CIFAR-10 dataset, which is widely used in NAS [16, 73].

5.1 Case Studies

5.1.1 Analyzing NAS-Bench-101

In this case study, we collaborated with expert E_1 to show how ArchExplorer helps summarize design principles from a large architecture space, NAS-Bench-101 [73], which contains 423,624 architectures. The number of layers in each architecture is limited to five, and the

number of connections between layers is limited to nine. The layers are chosen from: 3×3 convolution, 1×1 convolution, and 3×3 max-pooling. Two example architectures are shown in Fig. 1G.

Overview. E_1 started the analysis by examining the global relationships between the clusters in Fig. 1(a). He found that the clusters at the top had more dark green circles and shorter red arcs (max-pooling layers) than those at the bottom-left. This indicates that the architectures using fewer max-pooling layers have better performance. In particular, he saw that clusters A, B, D contained more well-performing architectures. E_1 decided to analyze them in detail.

Analyzing architectures without max-pooling layers. E_1 first examined cluster A that contained the architectures without max-pooling layers. Since the accuracy variance of this cluster is large, he zoomed into it for further analysis. The well-performing architectures in cluster A are mostly located in three sub-clusters (Fig. 1(b)). By examining the structure glyphs, he found that sub-cluster H used four convolution layers and the other two used five convolution layers. However, they have comparable numbers of well-performing architectures. This attracted his attention because more convolution layers usually lead to better performance. E_1 further explained, "Architectures with more convolution layers have a larger number of trainable parameters and thus a larger model capacity. Typically, a larger model capacity can better fit the data and achieve better performance." Since 40 million approaches the upper limit of trainable parameters for architectures with four 3×3 convolution layers, he selected such architectures from the three sub-clusters by using the "params" dimension of the PCP (the dashed ellipse in Fig. 1E) to figure out why this phenomenon occurred. Then he sorted them by accuracy. By comparing the individual architectures in Fig. 1F, E_1 found that the architectures in H had more skip-connections (the rows with orange borders). By examining their structures, he found that they resembled the structure of DenseNet [22], which had dense skip-connections between layers (the thick black lines in Fig. 1G). He commented that dense skip-connections combine the outputs of multiple previous layers and thus strengthen feature propagation [22]. This leads to better performance. Thus, he suggested:

• *Principle 1: dense skip-connections are beneficial to accuracy.*

Analyzing architectures with max-pooling layers. E_1 then continued to analyze other well-performing architectures in clusters B and D (Fig. 1(a)). The short red arcs in the summary glyphs and the structure glyphs reveal that they only use one max-pooling layer. A nearby cluster C caught his attention. In this cluster, the red arcs of the summary glyphs have the same lengths as those in B and D, but the architectures in this cluster have much lower overall accuracy. By comparing their structure glyphs, he found that the major difference between the architectures in cluster C and clusters B and D was the position of the max-pooling layer. In B and D, the architectures have their max-pooling layers at the front (cluster B) or in the middle (cluster D) of the structure glyphs. While in cluster C, the architectures have their max-pooling layers at the end.

To understand the effect caused by the position of the max-pooling layer, we collaborated with E_1 and analyzed the activation map differences between the corresponding architectures in clusters B and C. We randomly selected two architectures a_1 (accuracy: 94.6%) and a_2 (accuracy: 93.8%) from clusters B and C and trained them on CIFAR-10. We fed each image into the trained model and obtained the feature map with the largest response before the fully connected layer as its activation map [18]. By analyzing a set of activation maps of images, each of which is predicted differently by a_1 and a_2, he found that the response areas generated by a_1 were usually more precise than those by a_2. This indicates that architectures with the max-pooling layers at the end probably introduce irrelevant information for prediction, leading to more misclassifications. For example, in Fig. 7, a_2 misclassified a bird as a frog since it mistook the larger green background as a grassland. We further verified this by checking the activation maps of 100 random images. The results showed that although a_1 and a_2 both learned to focus on the objects in almost all images (99%), a_2 was more likely to generate imprecise response areas (49%) than a_1 (14%). Following a similar analysis, we also found that the response areas generated by the architectures with the max-pooling layer in the middle were more precise than those with the max-pooling layer at the end. Based on this observation, E_1 commented, "The architectures with max-pooling layers at the end enlarge the response areas and tend to introduce information irrelevant to prediction. This may lead to more misclassifications." Therefore, he concluded:

• *Principle 2: max-pooling layers should probably not appear at the end of the architecture.*

Comparing architectures without and with max-pooling layers. After analyzing the architecture clusters without and with max-pooling layers separately, E_1 was interested in why the well-performing architectures did not use many max-pooling layers. He randomly selected and trained two architectures a_3 (accuracy: 95.1%) and a_4 (accuracy: 93.9%) with zero and one max-pooling layer, respectively. With a similar analysis as previously described, he found that architectures with max-pooling layers often introduced irrelevant information into the prediction process and thus led to more misclassifications. For example, in Fig. 8, a_4 misclassified a ship as an airplane since it mistook the horizontal black bars in the background as the wings of an airplane. For validation, 100 random images were also investigated to examine their activation map differences. The results suggested that both a_3 and a_4 can learn the key part of the object in nearly 90% of the images, but a_4 was more likely to learn some interfering parts (76%) than a_3 (34%). From this analysis, E_1 concluded:

• *Principle 3: the max-pooling layers probably downgrade accuracy.*

5.1.2 Analyzing NAS-Bench-201

We collaborated with expert E_2 to demonstrates how to design an architecture with better generalization ability based on the design principles summarized from NAS-Bench-201 [16]. This space consists of 15,625 architectures. Each architecture contains six layers chosen from: 1×1 convolution, 3×3 convolution, 3×3 average-pooling, identity, and zeroize. An example architecture is shown in Fig. 10(b).

Summarizing design principles. E_2 started the analysis from the architecture visualization (Fig. 9(a)). By examining the summary glyphs, he found that architectures were clustered based on the number of zeroize layers (grey arcs). Cluster A has the largest number of well-performing architectures, where architectures do not use any zeroize layers. This met his expectation because the architectures with such layers have less trainable parameters and thus smaller model capacity. This leads to performance degradation.

In cluster A, a region with many darkest green circles (Fig. 9B)

Fig. 7: The activation map differences between architectures with the max-pooling layer at the front (a_1) and at the end (a_2). a_1 generated a more precise response area on the bird than a_2.

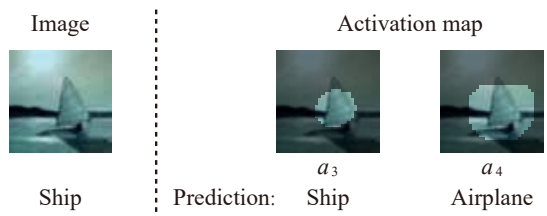

Fig. 8: The activation map differences between architectures with zero (a_3) and one (a_4) max-pooling layer. a_3 generated a more precise response area on the ship than a_4.

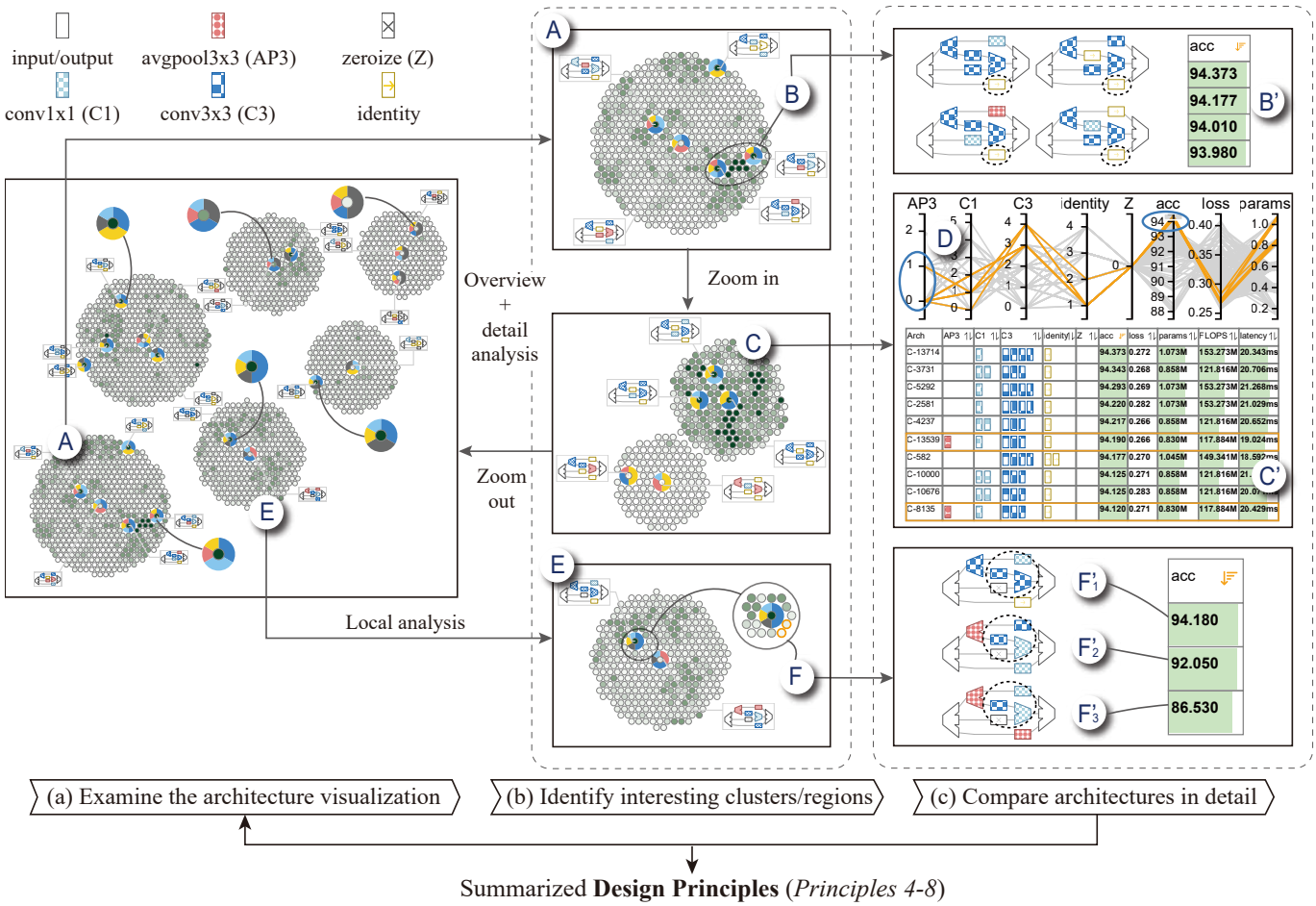

Fig. 9: The analysis workflow of NAS-Bench-201. The dashed ellipses mark the similar parts of the shown architectures.

aroused his interest. This indicates that these well-performing architectures have similar structures. Thus, he used the lasso to select these architectures for detailed examination. E_2 noticed that all these architectures had an identity layer connecting the input and the output (dashed ellipses in Fig. 9B'). He commented that such an identity layer could address the vanishing gradient problem in model training because it provides an alternative path for the gradient flow in back-propagation. This is consistent with the design of ResNet [19]. Therefore, he suggested:

• *Principle 4: the identity layer connecting the input and the output is beneficial to the performance.*

Due to the large accuracy variance in cluster A, E_2 zoomed into this cluster for detailed examination. E_2 found that sub-cluster C contained the most well-performing architectures. He selected it and filtered the architectures with the highest accuracy by using the "acc" dimension of the PCP. With a further examination of the PCP (Fig. 9D), E_2 found that most well-performing architectures did not contain any 3×3 average-pooling layer, and a few of them used one such layer. For verification, E_2 also examined the architectures with the top 10 accuracy in the table (Fig. 9C'). Only two out of them use one average-pooling layer (the rows with orange borders), while the others do not use it. He commented that this followed the results of recent NAS methods [55, 63], where average-pooling layers seldom appeared in the final searched architectures. Furthermore, we conducted a statistical test to compare the accuracy between architectures with and without average-pooling layers in NAS-Bench-201. The result showed that the accuracy of the architectures with an average-pooling layer was significantly lower than those without average-pooling layers ($p < 0.001$). This indicates that:

• *Principle 5: average-pooling layers probably downgrade performance.*

Following a similar analysis (the detailed analysis can be found in supplemental material), E_2 concluded other three design principles:

• *Principle 6: using multiple* 3×3 *convolution layers in one path improves model performance.*

• *Principle 7: using multiple paths containing* 3×3 *convolution layers improves model performance.*

• *Principle 8: having two or more paths without a convolution layer downgrades model performance.*

Local analysis of the performance difference. Next, E_2 zoomed back to the overview and briefly examined the well-performing architectures in other clusters. These architectures are scattered in different locations (Fig. 9(a)). However, some of their adjacent architectures can have much lower accuracy. For example, in cluster E, there is a representative architecture with high accuracy (Fig. 9F) and a few adjacent ones with low accuracy. Typically, neighboring architectures have similar structures and perform similarly. To figure out the reason for the difference, E_2 analyzed them in context. He selected this well-performing architecture (F'_1) and two adjacent poor-performing ones (the circles with orange borders, F'_2–F'_3) for comparison (Fig. 9F). By examining their structures, he found that F'_1 followed *Principles 4–8* and had the highest accuracy. F'_2 followed *Principles 7 and 8*, and its accuracy was lower than that of F'_1 by 2.13%. F'_3 only followed *Principle 8*, and its accuracy was lower than that of F'_1 by 7.65%. The expert appreciated the capability of ArchExplorer to visually convey the performance difference among adjacent architectures. He further commented that the easy finding of such differences and the associated visual explanations help him summarize design principles in a more detailed manner.

Designing an architecture with better generalization ability. Based on the above design principles, we collaborated with E_2 to manually

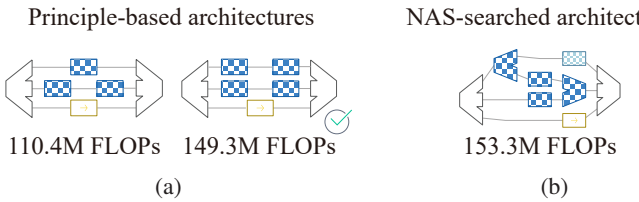

Fig. 10: Architecture comparison: (a) the principle-based architectures; (b) the NAS-searched architecture with the highest accuracy.

Table 1: Comparison of the generalization ability between the NAS-searched architecture and the principle-based architecture.

Dataset	# classes	NAS-searched	Principle-based
Aircraft	41	85.3%	(+0.0%) 85.3%
Cars	196	74.2%	(+1.3%) 75.5%
Covid	4	95.0%	(+0.0%) 95.0%
DTD	47	56.6%	(+2.3%) 58.9%
GTSRB	43	97.4%	(+0.1%) 97.5%
Blood Cells	4	90.6%	(+3.7%) 94.3%
Scene	6	91.4%	(+0.2%) 91.6%
Average		84.4%	(+1.0%) 85.4%

design an architecture to evaluate the effectiveness of the summarized design principles. For a fair comparison, he only utilized the layer candidates in NAS-Bench-201. E_2 proposed the two simplest architectures that followed all principles discovered from NAS-Bench-201 (Fig. 10(a)). Each has an identity layer (*Principle 4*) and does not contain average-pooling layers (*Principle 5*). Besides, it has two paths with 3 convolution layers, and at least one of them contains multiple 3×3 convolution layers (*Principles 6 and 7*). The architecture design also followed *Principle 8* by containing only one path without a convolution layer. Generally, larger FLOPs lead to better performance. Since the best-performing architecture searched by the recent NAS methods on CIFAR-10 has 153.3M FLOPs [63, 70] (Fig. 10(b)), he finally selected the one having the largest number of FLOPs (149.3M).

The generalization ability of the selected principle-based architecture and the best-performing NAS-searched one was evaluated by comparing the performance on a set of image classification datasets. These datasets are from the publicly available Kaggle dataset [27]: *Aircraft*, *Cars*, *Covid*, *DTD*, *GTSRB*, *Blood Cells*, and *Scene*. They cover different types of images, including real-world objects, medical images, textures, and scenes. The number of categories in these datasets varies from a few to more than a hundred. Table 1 shows the accuracy comparison between the NAS-searched architecture (column "NAS-searched") and the principle-based architecture (column "Principle-based"). We found that the principle-based architecture achieved better or comparable accuracy on all datasets. It had an average accuracy improvement of **1.0%**. This demonstrated the effectiveness of the summarized design principles for designing architectures with better generalization ability.

5.2 Post-Analysis

Two experiments were conducted to demonstrate the effectiveness of the design principles in improving the search efficiency of NAS.

Experimental settings. We selected a state-of-the-art NAS method, **LaNAS** [63], as our baseline. Then a **hybrid method** was implemented, which updated LaNAS to reflect *Principles 1–8* by discarding the violating architectures in its search process.

In the experiments, we employed two widely-used architecture spaces, NASNet [63] and NAS-Bench-301 [56]. In NASNet, each architecture contains ten layers chosen from four candidates: 3×3 max-pooling, 3×3 depth-separable convolution, 5×5 depth-separable convolution, and identity. In NAS-Bench-301, each architecture contains eight layers chosen from seven candidates, including 3×3 average-pooling, 3×3 dilated convolution, 5×5 dilated convolution, and the four candidates in NASNet.

The computation cost of a search process was evaluated by the total GPU hours for searching and training the architectures. It is prohibitive to train each searched architecture in NASNet from scratch to full convergence (about 60 GPU hours for each architecture and nearly 50,000 GPU hours in total for a search process). Following the recent research [49], we trained each architecture for 20 epochs on CIFAR-10. In NAS-Bench-301, we use the accuracy provided in the dataset. Since we did not need to train the architectures in this space, the GPU hours for training the searched architectures in this search were estimated by multiplying the number of searched architectures and the GPU hours for training one architecture. Here, the GPU hours for training one architecture were estimated by averaging the training time of ten randomly-sampled architectures in this space.

Table 2: Comparison of the search space, computation cost and accuracy on NASNet and NAS-Bench-301.

Dataset	Method	# archs.	GPU hours	Accuracy
NASNet	LaNAS	800	1,635	97.99%
	Hybrid	400	822	98.10%
NAS-301	LaNAS	2,000	3,019	94.83%
	Hybrid	1,000	1,510	94.83%

archs. refers to the number of searched architectures.

Results. Table 2 compares the search space, computation cost, and accuracy between LaNAS and the hybrid method with design principles on NASNet and NAS-Bench-301. Compared with LaNAS, the hybrid method reduced the search space and computation cost by around 50% while achieving at least the same accuracy on both datasets.

Analysis. To identify the reason for the computation cost reduction, we first analyzed the searched architectures found by LaNAS (without integrating any design principle) on NAS-Bench-301 (Fig. 11(a)). To better understand the search process of LaNAS, we analyzed the searched architectures at different iterations of the search process. We utilized a scented widget to highlight the architectures at different iterations in the architecture visualization. The analysis started from the architectures searched in the first 25% iterations of LaNAS. We found that these architectures randomly came from different clusters with different numbers of pooling layers. We further examined the architectures in the 25%–50%, 50%–75%, and the last 25% iterations. It was found that architectures without a pooling layer appeared more frequently as the search went on. This indicates that LaNAS gradually "learns" a few design principles for searching well-performing architectures, such as the preference of using fewer or even no pooling layers in the search process (*Principles 3 and 5*). To discover the common properties of the well-performing architectures, we selected them by using the "acc" dimension of the PCP (Fig. 11B). We found that they had one to three identity layers (Fig. 11A). By examining their structures, we identified that they had identity layers connecting the input and the output (Fig. 11(c)), which followed *Principle 4*. We then counted the occurrence of the architectures with such property at different iterations. These architectures appeared more often in the last 25% of the searched architectures than in the previously ones (50.2% *vs*. 29.8%), showing that LaNAS also learned such knowledge in the search process.

Second, after understanding the working mechanism of the search process of LaNAS, we briefly elaborated on why design principles could reduce the computation cost. By integrating the design principles, LaNAS filters out the architectures that violate any design principles and thus quickly focuses on searching among the well-performing architectures. This reduces the search space (2,000→1,000) and computation cost while also keeping the accuracy of the searched architectures.

6 Discussion and Future Work

We conducted a three-hour demo session with the experts. Overall, they appreciated the usefulness and effectiveness of ArchExplorer in summarizing design principles and searching for better-performing architectures. They especially liked the combination of easy-to-use and familiar visualization techniques, such as circle packing and pie-chart-based summary glyphs. It allows them to find the architectures of interest more quickly and thus focus more on the analysis tasks. Based

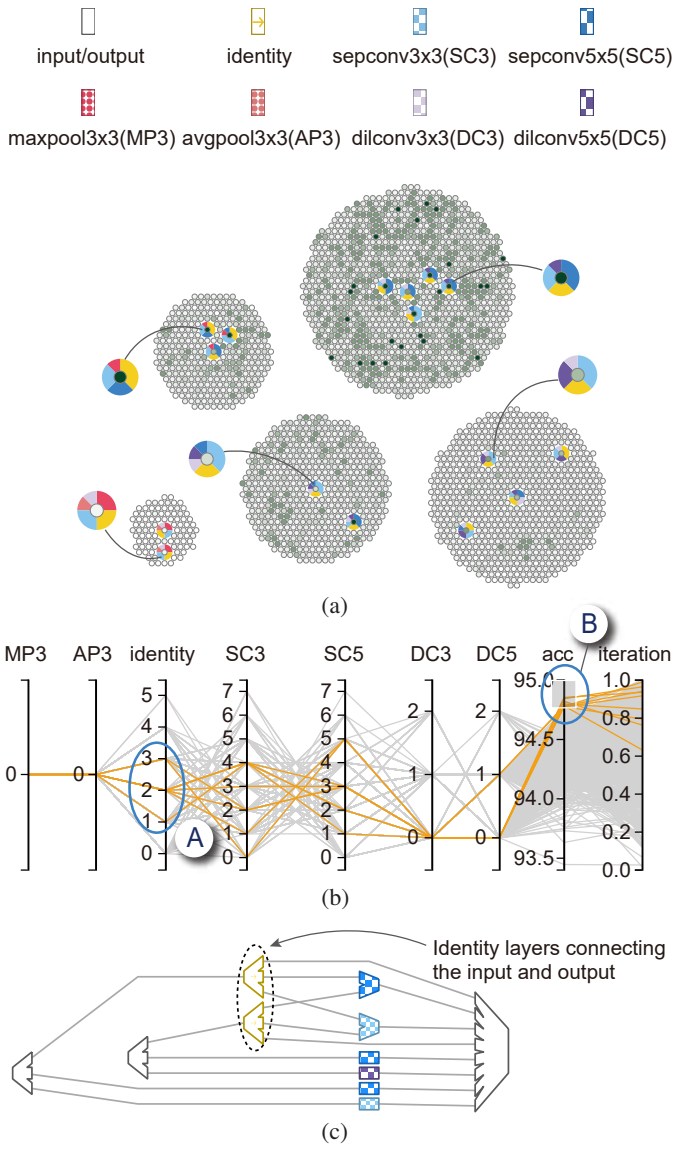

Fig. 11: Analyzing the searched architectures by LaNAS on NAS-Bench-301: (a) overview of the architecture clusters; (b) filtering the well-performing architectures; (c) an example of the well-performing architecture that has identity layers connecting the input and output.

on a comprehensive analysis, the experts can propose and validate design principles. They also pointed out some limitations that might lead to future research directions.

Providing more exploration guides. During the collaboration with the experts, we obtained two main needs for acquiring information more easily. First, they required to automatically summarize the characteristics of an architecture cluster and generate a meaningful label (*e.g.*, a short phrase) for it. For example, in Fig. 1(a), the desired labels of cluster C can be "using one max-pooling layer and four convolution layers" and "max-pooling layer appearing at the end of the architecture." Such labels can reduce the efforts in identifying the clusters of interest. In addition, this feature will benefit practitioners with average domain knowledge in the model designing process. As commented by experts, the interpretation of current cluster characteristics requires some expertise in the target architecture space. Thus, an interesting avenue is how to leverage the natural language processing techniques, such as GPT-3 [4], to generate meaningful labels for architecture clusters and intuitively illustrate them in the architecture visualization. Second, they hoped to visually search the architectures with specific structure components. For example, the experts are interested in how performance changes when replacing an identity layer with a 1×1 convolution layer in an architecture of interest. Therefore, another promising future work is to support a structure-based visual query for searching architectures.

Recommending candidate design principles. The capability of ArchExplorer in facilitating experts to summarize design principles is demonstrated in the case studies. The experts appreciated this capability because design principles are useful for designing a better-performing architecture. However, the current analysis workflow requires them to explore the architecture space, identify the architectures of interest, and then compare them in context. This takes some time for machine learning experts and even longer time for junior model developers. To accelerate the analysis process, the experts required the tool to automatically recommend candidate design principles. Based on the recommendation, they can visually analyze the associated architectures and verify the validity of the candidate design principles. Thus, in the future, we are interested in developing an efficient algorithm for recommending candidate design principles and tightly integrating it with our tool for iteratively verifying these candidates.

Integrating into the search process of a NAS algorithm. The experts also pointed out that integrating ArchExplorer into the search process of a NAS algorithm would be very useful and effective to reduce the computation cost. This integration opens up the possibility for incrementally integrating the design principles discovered at previous search iterations into the next iteration of the NAS algorithm. To facilitate such an incremental integration, ArchExplorer needs two major augmentations. First, we need to develop an incremental hierarchical clustering algorithm for effectively handling newly searched architectures at each iteration. Second, a set of interactions are required for incrementally and efficiently integrating the summarized design principles into the search process. For example, we can automatically convert the design principles into a set of constraints and then allow users to interactively refine them based on their search purposes. The refined constraints are utilized by the next search iteration for fast convergence.

Analyzing broader neural architecture spaces. In ArchExplorer, we focus on analyzing the influence of the structure on model performance. In general, the performance of a neural network architecture is also influenced by other model-related factors, such as training hyperparameters (learning rates, batch sizes, etc.), training procedure, and data distribution [11, 23, 71]. Thus, it would be useful to jointly consider these factors in ArchExplorer. To this end, there are some necessary extensions for ArchExplorer. For example, it is worth exploring how to tightly combine ArchExplorer with existing visual analytics works on analyzing data-distribution-related issues for summarizing the design principles from both structure and data perspectives.

7 Conclusion

In this paper, we have developed ArchExplorer, a visual analysis method for understanding a neural architecture space and summarizing design principles. The neural network architectures are represented by directed acyclic graphs, and graph edit distance is employed to model the similarity relationships between them. We formulate the pairwise distance calculation between architectures as an all-pairs shortest path problem and solve it with an accelerated Dijkstra algorithm. Based on the calculated distances, the architectures are then hierarchically clustered. A circle-packing-based architecture visualization has been developed to facilitate the interactive analysis of the architecture space. This visualization well conveys both the global relationships between clusters and the local neighborhoods of the architectures in each cluster. The effectiveness of ArchExplorer is demonstrated by two case studies, and the usefulness of the summarized design principles is verified by reducing the computation cost of a state-of-the-art NAS method.

Acknowledgments

This work was supported by National Key R&D Program of China under Grant 2020YFB2104100, the National Natural Science Foundation of China under grants U21A20469 and 61936002, grants from the Institute Guo Qiang, THUIBCS, and BLBCI, and in part by Tsinghua-Kuaishou Institute of Future Media Data. The authors would like to thank Weikai Yang, Chengjian Chen and Zhen Li for their valuable comments.

REFERENCES

[1] M. Abdel-Basset, G. Manogaran, H. Rashad, and A. N. H. Zaied. A comprehensive review of quadratic assignment problem: Variants, hybrids and applications. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–24, 2018.

[2] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *Proceedings of the International Conference on Pattern Recognition Applications and Methods*, pages 271–278, 2015.

[3] P. Arora, Deepali, and S. Varshney. Analysis of K-Means and K-Medoids algorithm for big data. *Procedia Computer Science*, 78:507–512, 2016.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1877–1901, 2020.

[5] A. Bäuerle, C. van Onzenoodt, and T. Ropinski. Net2Vis – A visual grammar for automatically generating publication-tailored CNN architecture visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 27(6):2980–2991, 2021.

[6] D. Cashman, S. R. Humayoun, F. Heimerl, K. Park, S. Das, J. Thompson, B. Saket, A. Mosca, J. Stasko, A. Endert, M. Gleicher, and R. Chang. A user-based visual analytics workflow for exploratory model analysis. *Computer Graphics Forum*, 38(3):185–199, 2019.

[7] D. Cashman, A. Perer, R. Chang, and H. Strobelt. Ablate, variate, and contemplate: Visual analytics for discovering neural architectures. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):863–873, 2020.

[8] H.-C. Chang and L.-C. Wang. A simple proof of Thue's theorem on circle packing. *arXiv preprint arXiv:1009.4322*, 2010.

[9] A. Chatzimparmpas, R. M. Martins, K. Kucher, and A. Kerren. VisEvol: Visual analytics to support hyperparameter search through evolutionary optimization. *Computer Graphics Forum*, 40(3):201–214, 2021.

[10] C. Chen, Z. Wang, J. Wu, X. Wang, L.-Z. Guo, Y.-F. Li, and S. Liu. Interactive graph construction for graph-based semi-supervised learning. *IEEE Transactions on Visualization and Computer Graphics*, 27(9):3701–3716, 2021.

[11] C. Chen, J. Yuan, Y. Lu, Y. Liu, H. Su, S. Yuan, and S. Liu. OoDAnalyzer: Interactive analysis of out-of-distribution samples. *IEEE Transactions on Visualization and Computer Graphics*, 27(7):3335–3349, 2021.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[13] S. Das, S. Xu, M. Gleicher, R. Chang, and A. Endert. QUESTO: Interactive construction of objective functions for classification tasks. *Computer Graphics Forum*, 39(3):153–165, 2020.

[14] J. F. DeRose, J. Wang, and M. Berger. Attention Flows: Analyzing and comparing attention mechanisms in language models. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1160–1170, 2021.

[15] X. Dong and Y. Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3681–3690, 2019.

[16] X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the International Conference on Learning Representations*, 2020.

[17] T. Elsken, J. H. Metzen, F. Hutter, et al. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[20] X. He, K. Zhao, and X. Chu. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[21] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics*, 25(8):2674–2693, 2019.

[22] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2261–2269, 2017.

[23] F. Isensee, P. F. Jaeger, S. A. Kohl, J. Petersen, and K. H. Maier-Hein. nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature methods*, 18(2):203–211, 2021.

[24] T. Jaunet, C. Kervadec, R. Vuillemot, G. Antipov, M. Baccouche, and C. Wolf. VisQA: X-raying vision and language reasoning in transformers. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):976–986, 2022.

[25] H. Jin, Q. Song, and X. Hu. Auto-Keras: An efficient neural architecture search system. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.

[26] J. Johansson and C. Forsell. Evaluation of parallel coordinates: Overview, categorization and guidelines for future research. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):579–588, 2016.

[27] Kaggle. Kaggle datasets. `https://www.kaggle.com/datasets`, 2022. Last accessed 2022-03-31.

[28] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau. ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2018.

[29] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[30] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 2020–2029, 2018.

[31] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, 2020.

[32] K. Koffka. *Principles of Gestalt psychology*. Routledge, 2013.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[34] Z. Li, X. Wang, W. Yang, J. Wu, Z. Zhang, Z. Liu, M. Sun, H. Zhang, and S. Liu. A unified understanding of deep nlp models for text classification. *IEEE Transactions on Visualization and Computer Graphics*, 2022. to be published, doi: 10.1109/TVCG.2022.3184186.

[35] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.

[36] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu. Analyzing the training processes of deep generative models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):77–87, 2018.

[37] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, 2017.

[38] S. Liu, J. Xiao, J. Liu, X. Wang, J. Wu, and J. Zhu. Visual diagnosis of tree boosting methods. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):163–173, 2018.

[39] Y. Ma, A. Fan, J. He, A. R. Nelakurthi, and R. Maciejewski. A visual analytics framework for explaining and diagnosing transfer learning processes. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1385–1395, 2021.

[40] Y. Meng, H. Zhang, M. Liu, and S. Liu. Clutter-aware label layout. In *Proceedings of the IEEE Pacific Visualization Symposium*, pages 207–214, 2015.

[41] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu. Understanding hidden memories of recurrent neural networks. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology*, pages 13–24, 2017.

[42] R. Moller. Path planning using hardware time delays. *IEEE Transactions on Robotics and Automation*, 15(3):588–592, 1999.

[43] S. Murugesan, S. Malik, F. Du, E. Koh, and T. M. Lai. DeepCompare: Visual and interactive comparison of deep learning model performance. *IEEE Computer Graphics and Applications*, 39(5):47–59, 2019.

[44] V. Nguyen, T. Le, M. Yamada, and M. A. Osborne. Optimal transport kernels for sequential and parallel neural architecture search. In *Proceedings of the International Conference on Machine Learning*, pages 8084–8095, 2021.

[45] X. Ning, Y. Zheng, T. Zhao, Y. Wang, and H. Yang. A generic graph-based neural architecture encoding scheme for predictor-based NAS. In *Proceedings of the European Conference on Computer Vision*, pages 189–204, 2020.

[46] J. P. Ono, S. Castelo, R. Lopez, E. Bertini, J. Freire, and C. Silva. PipelineProfiler: A visual analytics tool for the exploration of AutoML pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):390–

400, 2021.

[47] H. Park, Y. Nam, J.-H. Kim, and J. Choo. HyperTendril: Visual analytics for user-driven hyperparameter optimization of deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1407–1416, 2021.

[48] H.-S. Park and C.-H. Jun. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36(2, Part 2):3336–3341, 2009.

[49] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár. Designing network design spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10425–10433, 2020.

[50] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, 2017.

[51] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):61–70, 2017.

[52] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009.

[53] D. Sacha, M. Kraus, D. A. Keim, and M. Chen. VIS4ML: An ontology for visual analytics assisted machine learning. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):385–395, 2019.

[54] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

[55] H. Shi, R. Pi, H. Xu, Z. Li, J. T. Kwok, and T. Zhang. Bridging the gap between sample-based and one-shot neural architecture search with BONAS. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1808–1819, 2020.

[56] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

[57] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):667–676, 2018.

[58] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 6105–6114, 2019.

[59] L. Van Der Maaten, E. Postma, J. Van den Herik, et al. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10(13):66–71, 2009.

[60] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, et al. FBNetV2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020.

[61] X. Wan, B. Ru, P. M. Esperança, and Z. Li. On redundancy and diversity in cell-based neural architecture search. In *International Conference on Learning Representations*, 2022.

[62] D. Wang, J. Andres, J. D. Weisz, E. Oduor, and C. Dugan. AutoDS: Towards human-centered automation of data science. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2021.

[63] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian. Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. to be published, doi: 10.1109/TPAMI.2021.3071343.

[64] Q. Wang, Y. Ming, Z. Jin, Q. Shen, D. Liu, M. J. Smith, K. Veeramachaneni, and H. Qu. ATMSeer: Increasing transparency and controllability in automated machine learning. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[65] Q. Wang, Z. Xu, Z. Chen, Y. Wang, S. Liu, and H. Qu. Visual analysis of discrimination in machine learning. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1470–1480, 2021.

[66] W. Wang, H. Wang, G. Dai, and H. Wang. Visualization of large hierarchical data by circle packing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 517—520, 2006.

[67] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng, and D. H. Chau. CNN Explainer: Learning convolutional neural networks with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1396–1406, 2021.

[68] D. K. I. Weidele, J. D. Weisz, E. Oduor, M. Muller, J. Andres, A. Gray, and D. Wang. AutoAIViz: Opening the blackbox of automated artificial intelligence with conditional parallel coordinates. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 308–312, 2020.

[69] C. White, W. Neiswanger, S. Nolen, and Y. Savani. A study on encodings for neural architecture search. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 20309–20319, 2020.

[70] J. Wu, X. Dai, D. Chen, Y. Chen, M. Liu, Y. Yu, Z. Wang, Z. Liu, M. Chen, and L. Yuan. Stronger NAS with weaker predictors. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1–14, 2021.

[71] W. Yang, Z. Li, M. Liu, Y. Lu, K. Cao, R. Maciejewski, and S. Liu. Diagnosing concept drift with visual analytics. In *IEEE Conference on Visual Analytics Science and Technology*, pages 12–23. Institute of Electrical and Electronics Engineers (IEEE), 2020.

[72] W. Yang, X. Ye, X. Zhang, L. Xiao, J. Xia, Z. Wang, J. Zhu, H. Pfister, and S. Liu. Diagnosing ensemble few-shot classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 28(9):3292–3306, 2022.

[73] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the International Conference on Machine Learning*, pages 7105–7114, 2019.

[74] J. Yuan, C. Chen, W. Yang, M. Liu, J. Xia, and S. Liu. A survey of visual analytics techniques for machine learning. *Computational Visual Media*, 7(1):3–36, 2021.

[75] J. Yuan, S. Xiang, J. Xia, L. Yu, and S. Liu. Evaluation of sampling methods for scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1720–1730, 2021.

[76] X. Zhao, Y. Wu, D. L. Lee, and W. Cui. iForest: Interpreting random forests via visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):407–416, 2019.

[77] B. Zoph, V. Vasudevan, J. Shlens, and Q. 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.