ON REDUNDANCY AND DIVERSITY IN CELL-BASED NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Searching for the architecture cells is a dominant paradigm in NAS. However, little attention has been devoted to the analysis of the cell-based search spaces even though it is highly important for the continual development of NAS. In this work, we conduct an empirical post-hoc analysis of architectures from the popular cell-based search spaces and find that the existing search spaces contain a high degree of redundancy: the architecture performance is minimally sensitive to changes at large parts of the cells, and universally adopted designs, like the explicit search for a reduction cell, significantly increase the complexities but have very limited impact on the performance. Across architectures found by a diverse set of search strategies, we consistently find that the parts of the cells that do matter for architecture performance often follow similar and simple patterns. By explicitly constraining cells to include these patterns, randomly sampled architectures can match or even outperform the state of the art. These findings cast doubts into our ability to discover truly novel architectures in the existing cell-based search spaces, and inspire our suggestions for improvement to guide future NAS research.

1 INTRODUCTION

Neural Architecture Search (NAS), which automates designs of neural networks by task, has seen enormous advancements since its invention. In particular, *cell-based* NAS has become an important technique in NAS research: in contrast to attempts that directly aim to design architectures at once and inspired by the classical manually-designed architectures like VGGNet and ResNet that feature repeated blocks, cell-based NAS similarly searches for repeated *cells* only and later stack them into full architectures. This simplification reduces the search space (but still highly complex), and allows easy transfer of architectures across different tasks/datasets, application scenarios and resources constraints (Elsken et al., 2019). Indeed, it has received an extraordinary amount of research attention: based on our preliminary survey, **almost 80%** of the papers (detailed in App C) proposing new NAS methods published in the top machine learning conferences (ICLR, ICML, NEURIPS) in the past year show at least one part of their major results in standard Differentiable Architecture Search (DARTS) cell-based spaces and/or highly related ones, and **approx. 60%** demonstrate results in such spaces *only*; it is fair to state that such cell-based spaces currently dominates.

However, the lagging understanding of these architectures and the search space itself stands in stark contrast with the volume and pace of new *search algorithms*. The literature on *understanding why this dominant search space works* and *comparing what different NAS methods have found* is much more limited in depth and scope, with most existing works typically focusing exclusively on a few search methods or highlighting high-level patterns only (Shu et al., 2020; Zela et al., 2020) We argue that strengthening such an understanding is crucial on multiple fronts, and the lack thereof is concerning: first, studying a diverse set of architectures enables us to discover patterns shared across the search space, the foundation of comparison amongst search methods that heavily influences the resultant performances (Yang et al., 2020a): if the search space itself is flawed, designing and iterating new search methods on it, which is typically computationally intensive, can be misleading and unproductive. Conversely, understanding any pitfalls of the existing search spaces informs us on how to design better search spaces in the future, which is critical in advancing the goal of NAS in finding novel and high-performing architectures, not only in conventional CNNs but also in emerging NAS paradigms such as transformers (which may also take the form of a cell-based design space). Second, opening the NAS black box enables us to distill the essence of the strong-performing NAS architectures beneath their surface of complexity. Unlike manually designed architectures

where usually designers attribute performance to specific designs, currently owing to the apparent complexity of the design space, the NAS architectures, while all discovered in a similar or identical search space, are often compared to in terms of final performance on a standard dataset only (e.g. CIFAR-10 test error). This could be problematic, as we do not necessarily understand *what* NAS has discovered that led to the purported improvements, and the metric itself is a poor one on which external factors such as hyperparameter settings, variations in training/data augmentation protocols and even just noise could exert a greater influence than the architectural design itself (Yang et al., 2020a). However, by linking performance to specific designs, we could more ascertain whether any performance differences stem from the architectures rather than the interfering factors.

We aim to address this problem by presenting a post-hoc analysis of the well-performing architectures produced by technically diverse search methods. Specifically, we utilise explainable machine learning tools to open the NAS black box by inspecting the good- and bad-performing architectures produced by a wide range of NAS search methods in the dominant DARTS search space. We find:

- Performances of architectures can often be disproportionately attributed to a small number of simple yet critical features that resemble *known* patterns in classical network designs;
- Many designs almost universally adopted simply contribute to complexity but not performance;
- The nominal complexity of the search spaces poorly reflects the actual diversity of the (high-performing) architectures discovered, the functional parts of which are often very similar despite the technical diversity in search methods and the seeming disparity in topology.

In fact, with few simple and human-interpretable constraints, almost *any* randomly sampled architecture can perform on par or exceed those produced by the state-of-the-art NAS methods over varying network sizes and datasets (CIFAR-10/IMAGENET). Ultimately, these findings prompt us to rethink the suitability of the current standard protocol in evaluating NAS and the capability to find truly novel architectures within the search space. We finally provide suggestions for prospective new search spaces inspired by these findings.

2 PRELIMINARIES

DARTS space (Fig 1) proposed by Liu et al. (2019), which is in turn inspired from Zoph et al. (2018); Pham et al. (2018), is the most influential search space in cell-based NAS: it takes a form of a Directed Acyclic Graph (DAG), which features 2 input (connected to the output of two immediately preceding cells), 1 output and 4 intermediate nodes. The *operation* $o^{(i,j)}$ on the (i,j)-th edge is selected from a set of candidate *primitives* $\mathbb{A}$ with size $K = 7$ (shown in Fig 1) to transform the input to the operation $x^{(i)}$. At each intermediate node, output from the operations on all its predecessors are aggregated: $x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$ and finally the `out` node concatenates all outputs from the intermediate nodes. As shown in Fig 1, the DARTS cell allows for up to 14 possible spots to place operations. However, to ensure that all intermediate nodes are connected in the computational graph, each intermediate node is constrained to have exactly 2 in-edges connected to it from 2 different preceding nodes. Lastly, it is conventional to search for two cells, namely *normal* α_n and *reduce* α_r cells simultaneously, with α_r placed at 1/3 and 2/3 of the total depth of the networks and α_n everywhere else. In total, there are $\prod_{k=1}^{4} \frac{7^2 k(k+1)}{2} \approx 10^9$ distinct cells without considering graph isomorphism, and since both α_n and α_r are required to fully specify an architecture, there exist approx. $(10^9)^2 = 10^{18}$ distinct architectures (Liu et al., 2019). Other cells commonly used in are almost invariably closely related to, and can be viewed as simplified or more complicated variants of the DARTS space. For example, NAS-Bench-201 (NB201) search space used in the NAS benchmark is similar to but simplified from the DARTS cell (detailed and analysed in App D). Other works have increased the number of intermediate nodes (Wang et al., 2021b), expanded the primitive pool $\mathbb{A}$ (Hundt et al., 2019) and/or relaxed other constraints (e.g. Shi et al. (2020) allow both incoming edges of the intermediate nodes to be from the same preceding nodes), but do not fundamentally differ from the DARTS space.

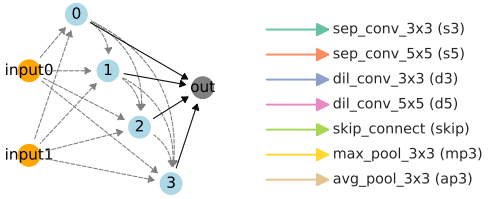


Figure 1: The DARTS cell. The solid black arrows denote the concatenation at output which is fixed. The gray dashed arrows denote the 14 potential operation locations. In a valid DARTS cell, 8 of which are filled by one out of the seven candidate primitives listed to the right, while the other 6 spots are disabled.

3 OPERATION-LEVEL ANALYSIS: REDUNDANCIES IN SEARCH SPACES

Cell-based NAS search space contains multiple sources of complexity (deciding both the specific wiring and operations on the edges; searching for 2 cells independently, etc.), each expanding the search space combinatorially. While this complexity is argued to be necessary for the search space to be expressive for good-performing novel architectures to be found, it is unknown whether *all* these sources of complexities are equally important, or whether the performance critically depend on some sub-features only. We argue that answering this question, and identifying such features, if they are indeed present, are highly significant: it enables us to separate the complexities that positively contribute to performances from those that do not, and subsequently would fundamentally affect the design of search methods. At individual architectures, it helps us understand what NAS has truly discovered, by removing confounding factors and focusing on the key aspects of the architectures.

For findings to be generally applicable, we aim to be not specific to any search method; this requires us to study a large set of architectures that are high-performing but technically diverse in terms of the search methods that produce them. Fortunately, the training set of NAS-Bench-301 (NB301) (Siems et al., 2020), which include >50,000 architecture-performance pairs in the DARTS space using a combination of random sampling and more than 10 state-of-the-art yet technically diverse methods (detailed in App. B.1), could be used for such an analysis: to build a surrogate model that accurately predicts architecture performance across the entire space. We primarily focus on the top-5% (or top-2,589) architectures of the training set since we overwhelmingly care about the good-performing architectures by definition in NAS, although as we will show, the main findings hold true also for architectures found by methods not covered by NB301 and for other search spaces like the NB201 space. As shown in Fig 2, the top architectures are well-spread out in the search space and are well-separated by search methods, seemingly suggesting that the architectures discovered by different methods are diverse in characteristics. Lastly, the worse-performing cells could also be of interest, as any features observed could be the ones we would actively like to avoid, and we analyse them in App. A.

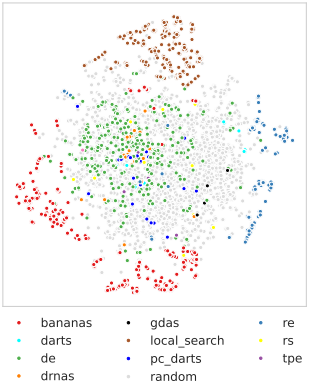


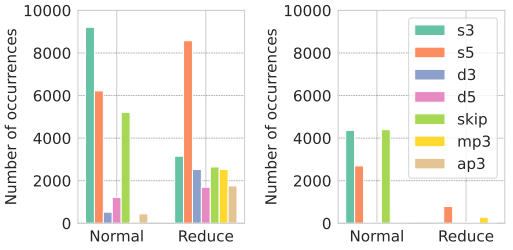
Figure 2: *The top archs provides a good coverage of the search space and are seemingly diverse*: the t-SNE plot of the top 5% archs (colored markers; grouped by different search methods. Details in App B.1) and randomly sampled archs in the DARTS search space (gray markers).

Operation Importance To untangle the influence of each part of the architecture, we introduce *Operation Importance (*OI*)*, which measures the incremental effect of the individual operations, which are the smallest possible features, to the overall performance. We quantify this via measuring the expected change in the performance by perturbing the type or wiring of the operation in question: considering an edge-attributed cell α with edge $e_{i,j}$ currently assigned with primitive o_k, then the operation importance of o_k on that edge of cell α is given by:

$$\text{OI}(\alpha, e_{i,j} := o_k) = \frac{1}{|\mathcal{N}(\alpha, e_{i,j} := o_k)|} \sum_{m=1}^{|\mathcal{N}(\alpha, e_{i,j}:=o_k)|} \Big[y(\alpha_m) \Big] - y(\alpha), \tag{1}$$

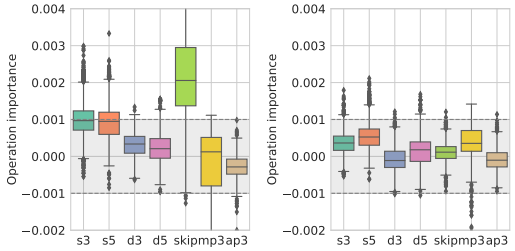
where $y(\alpha)$ denotes the validation accuracy or another appropriate performance metric of the fully-trained architecture induced by cell α. We are interested in measuring the importance of both the primitive choice and *where the operation is located relative to the entire cell*, and we use $\mathcal{N}(\alpha, e_{i,j} = o_k)$ to denote a set of neighbour cells that differ to α only with the edge in question assigned with another primitive: $e_{i,j} \in \mathbb{A} \setminus \{o_k\}$, or with the same primitive o_k but with one end node of $e_{i,j}$ being rewired to another node, subjected to any constraints in the search space. It is worth noting that OI is an instance of the *Permutation Feature Importance (*PFI*)*(Breiman, 2001; Fisher et al., 2019; Molnar, 2019). Given the categorical nature of the "features" in this case, we may enumerate all permutations on the edge in question instead of having to rely on random sampling as conventional PFI does. An important operation by Eq (1) would therefore be accorded with an OI with a large magnitude in either direction, whereas an irrelevant one would have a value of zero since altering it on expectation leads to no change in architecture performance.

We compute the OI of each operation of 2,589 architectures, and to circumvent the computational challenge of having to train all neighbour architectures of α (which are outside the NB301 training set) from scratch to compute $y(\cdot)$, we use the performance prediction $\tilde{y}(\cdot)$ from NB301. However, as



(a) *All* operations (b) *Important* operations

Figure 3: Distribution of (a) all and (b) important operations by the primitive types of the top-performing archiectures, organised by primitive type.



(a) Normal cells (b) Reduce cells

Figure 4: OI distribution in (a) normal and (b) reduce cells. The important operations are shown outside the gray shaded area.

we will show, we validate all key findings by actually training some architectures to ensure that they are not artefacts of the statistical surrogate (training protocols detailed in App. B.2).

Findings The most natural way to group the operations is by their primitive and cell (i.e. normal or reduce) types, and we show the main results in Figs. 3 and 4. In Fig. 3(b), we discretise the OI scores using the threshold 0.001 (0.1%), which is similar to the observed noise standard deviation of the better-performing architectures from NB301 which quantifies the expected variation in performance if an identical architecture is re-trained from scratch, to highlight the *important operations* with $|\text{OI}| \geq 0.001$: these the ones we could more confidently assert to affect the architecture performance beyond noise effects. We summarise the key findings below:

(#1) Only a fraction of operations is critical for good performance within cells: If all operations need to be fully specified (i.e., with both the primitive choice and the specific wiring determined) for good performance, perturbing any of them should have led to significant performance deterioration. However, this is clearly not the case in practice: comparing Fig. 3(a) and (b), we observe that only a fraction of the operations are important based on our definition.

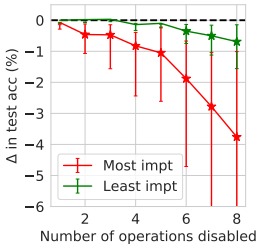


Figure 5: Ground-truth change in accuracy by successively disabling the most/least important ops, ordered by their OI. Medians and interquartile ranges shown; stars denote that the drop in accuracy is significant at $p \leq 0.01$ in the Wilcoxon signed-rank test.

To verify this directly beyond predicted performances, we randomly select 30 architectures from the top 5% training set. Within each cell, we sort their 16 operations by their OI in both ascending and descending orders. We then successively disable the operations by zeroising them, and train the resulting architectures with increasing number of operations disabled from scratch[1] until there are only half of the active operations remain (Fig. 5). The results largely confirms our findings and shows that the OI, although computed via predicted performance, is accurate in representing the ground-truth importance of the operations: on average, we need to disable 6 low-OI operations to see a statistically significant drop in performance, and almost half of the operations to match the effect of disabling just 2 high-OI ones. On the other hand, disabling the high-OI operations quickly reduce the performance and in some cases stall the training altogether: noting that the overall standard deviation of the entire NB301 training set is just approx. 0.8%, the drop in performance is quite dramatic.

(#2) Reduce cells are relatively unimportant for good performance: Searching independently for reduce cells scale the search space quadratically, but Fig. 3(b) shows that they contain much fewer important operations, and Fig. 4 shows that the OI distribution across all primitives are centered close to zero in reduce cells: both suggest that reduce cell is less important to the architecture performance. To verify this, we draw another 30 random architectures. For each of the them, we construct and train from scratch the original architecture and 4 derived ones, with (a) reduce cell set identical to normal cell, (b) reduce cell with all operations set to parameterless skip connections, (c) normal cell set identical to reduce cell and (d) normal cell with operations set to skip connections. As shown in Fig. 6: setting reduce cell to be identical to the normal cell leads to *no significant change in performance*, while the reverse is not true. A more extreme example is that while setting cells to be consisted of skip connections only is unsurprisingly sub-optimal in both cases, doing so on the reduce cells harms the performance much less – this suggests that while searching separately for reduce cells are well-motivated, the current design, which places much fewer reduce cells than normal cells in the overall architecture yet treats them equally in search might be a sub-optimal trade-off, and searching

[1]Note that we may not obtain NB301 performance prediction on these architectures, as NB301 requires all 16 operations to be enabled with valid primitives.

two separate cells may yield little benefits over the simple strategy of searching for one graph only and applying it on both normal and reduce cells.

(#3) Different primitives have vastly different importance profiles with many of them redundant: The set of candidate primitives $\mathbb{A}$ consists of operators that are known to be useful in manually-designed architectures, with the expectation that they should also be, to varying degrees, useful to NAS. However, this is clearly not the case: while it is already known that some primitives are favoured more by certain search algorithms (Zela et al., 2020), the observed discrepancy in the relative importance of the primitives is, in fact, more extreme: the normal cells (which are also the important cells by Finding 2) across the entire spectrum of good performing architectures overwhelmingly favour only 3 (separable convolutions & skip connection) out of 7 possible primitives (Fig. 3(a)). Even when the remaining 4 primitives are occasionally selected, they are almost never important (Fig. 3(b)) – this is also observed in Fig. 4(a) which shows that only they all have distributions of OI close to 0. As we will show later in Sec 4, we could essentially remove these primitives from $\mathbb{A}$ without impacting the performances. Even within the 3 favoured primitives, there is a significant amount of variation. First, comparing Figs 1(a) and (b), `skips`, when present in good architectures, are very likely to be important. We also note that the distribution of OI of `skip` has a higher variance – these suggest that the performance of an architecture is highly sensitive towards the specific locations and patterns of skip connections, a detailed study of which we defer to Sec. 4. On the other hand, while both separable convolutions (`s3` and `s5`) are highly over-represented in the good-performing architectures, it seems that they are less important on an operation level than `skip`. A possible explanation is that while their presence is required for good performance, their exact locations in the cell matter less which we again verify in Sec. 4.

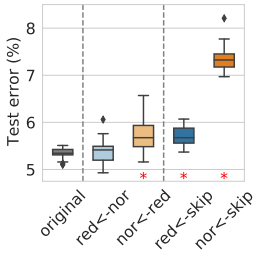


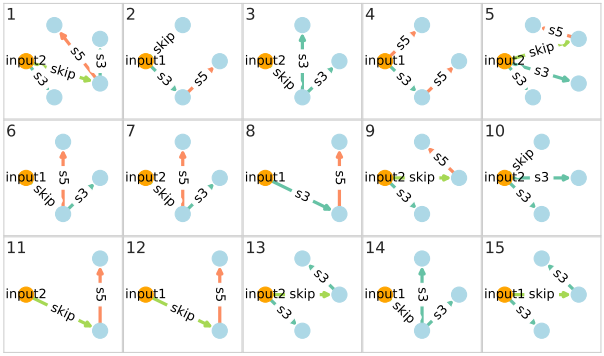
Figure 6: Ground-truth test errors (i.e. not predicted by NB301) of the original archs (`original`), archs with reduce cells set identical to their normal cells (`red<-nor`)/normal cells set identical to their reduce cells (`nor<-red`) and archs with normal/reduce cells fully replaced by skip connections (`nor<-skip`/`red<-skip`). $*$ denotes that the performance distribution significantly differs from `original` at $p \leq 0.01$ in the Wilcoxon signed-rank test.

Discussions The operation-level findings essentially confirm that both the search space and the cells are rampant with various redundancies that increase the search complexity but do not actually contribute as much to the performance, and good performance most often does not depend on an entire cell but a few key features and primitives in both individual cells and in the search space. This clearly shows that the search space design can be further optimised, but consequently, many beliefs often ingrained in existing search methods can also be sub-optimal or unnecessary. For example, barring a few exceptions (and none to our knowledge in the standard cell-based design space) (Xie et al., 2019a; You et al., 2020; Ru et al., 2020), the overwhelming majority of the current approaches aims to search for a single, fully deterministic architecture over a huge search space, and this often results in high-dimensional vector encoding of the cells (e.g. the path encoding of DARTS cell in White et al. (2021) is $> 10^4$ dimensions without truncation); this affects the performance in general (White et al., 2020a) and impedes the applications of methods that suffer from curse of dimensionality, such as Gaussian Processes, in particular. However, exact encoding could be in fact unnecessary if good performance simply hinges upon only a few specific key designs while the rest does not matter as much; it might make more sense to find relevant low-dimensional compressed representations such as approximate encoding instead.

4 Subgraph-level Analysis: Are we truly finding novel cells?

Sec 3 demonstrates the *presence* of the critical sub-features within good performing architectures; in this section we aim to find what they actually are and whether there are commonalities amongst the architectures found by technically diverse methods. Towards this goal, operation-level analysis is insufficient as the performance of neural networks also depends on the architecture topology and graph properties of the wiring between the operations (Xie et al., 2019a; Ru et al., 2020).

Frequent Subgraph Mining (FSM) FSM aims to "to extract all the frequent subgraphs, in a given data set, whose occurrence counts are above a specified threshold" (Jiang et al., 2013). This is immensely useful for our use-case, as any frequent subgraphs mined on the architectures represented by DAGs would naturally represent the interesting recurring structures in the good-performing



No.	# nodes	# edges	Support in impt	Support in ref	Ratio
1	5	4	0.0815	0.000386	231
2	4	3	0.0672	0.000772	87.0
3	4	3	0.0579	0.000772	75.0
4	4	3	0.0556	0.000772	72.0
5	5	4	0.0745	0.00116	64.3
6	4	3	0.219	0.00425	51.6
7	4	3	0.239	0.00579	41.3
8	3	2	0.215	0.00966	22.2
9	4	3	0.188	0.00888	21.3
10	4	3	0.103	0.00541	19.1
11	3	2	0.572	0.0339	16.8
12	3	2	0.336	0.0220	15.2
13	4	3	0.141	0.00927	15.2
14	4	3	0.0560	0.00386	14.5
15	4	3	0.0734	0.00541	13.6

Figure 7 & Table 1: Frequent subgraphs in the good-performing architectures ranked by ratio of supports between the important subgraphs and the reference and properties of the discovered frequent subgraphs. Almost all subgraphs feature `skip` primitive links with additional connections with 1 or more separable convolutions and neither dilated convolutions nor other parameterless operations.

architectures, and subgraphs are also widely used for generating explanations (Ying et al., 2019b). In our specific case, we 1) convert the computing graphs corresponding to the topology of the same set of top-performing architectures in Sec. 3 into DAGs, 2) within each DAG, we retain only the important operations defined in Sec 3 and 3) run an adapted version of the gSpan algorithm (Yan & Han, 2002) for DAGs on the set of all architecture cell graphs $\{G_1, ..., G_T\}$ to identify a set of the most frequent subgraphs $\mathcal{G}^f = \{g_1^f, ..., g_M^f\}$ where each subgraph must have a minimum *support* $S_{(\cdot)}$ of $\sigma = 0.05$:

$$S_{g_i^f} = \frac{|\delta(g_i^f)|}{T} \geq \sigma \; \forall g_i^f \in \mathcal{G}^f, \text{ where } \delta(g_i^f) = \{G_j | g_i^f \subseteq G_j\}_{j=1}^T. \tag{2}$$

One caveat with the above metric is that it favours simpler subgraphs, which naturally are more likely to be present in a graph by random sampling than more complicated subgraphs. To account for this bias, we use the same DAG representation of all architectures used earlier, but instead of retaining the important operations, we retain k_i *randomly* sampled operations in each dag to build a *reference set*, where k_i is the number of important operations present in architecture i. We then run gSpan again to obtain the supports of the reference frequent subgraphs $\mathcal{G}^c = \{g_1^c, ..., g_N^c\}$. The ratio of the two supports quantifies the amount of over- or under-representation of subgraphs over their "natural" level of occurrence and corrects the aforementioned bias.

Findings *(#4) Functional parts of many good-performing architectures are structurally similar to each other and to elements of classical architectures.* We show the top subgraphs in terms of the ratio of supports in Fig. 7, and an immediate insight is that the top frequent subgraphs representing the common traits across the entire good-performing region of the search space are highly over-represented over reference and *non-diverse*: almost all subgraphs can be characterised with skip connections forming residual links between one or both input nodes with an operation node, combined with different number and/or sizes of separable convolutions. In fact, we find that this ResNet-style residual link is present in 98.5% (or 2,815) of the 2,859 top architectures (as a comparison, if we sample randomly, only approximately half of the architectures are expected to contain this feature). In fact, with reference to Fig 8, the residual links *drive* the importance of `skip` in Fig 3 – this suggests skip connections do not just benefit optimisation of NAS supernets but also actively contribute to generalisation if they posit as residual connections. The propensity of certain NAS methods into collapsing into cells almost entirely consisting of `skip` is well-known with many possible explanations and remedies, but here we provide an alternative perspective independent of search methods: more fundamentally, `skip` is the only primitive whose exact position greatly impacts the performances in both positive and negative directions and thus it is more difficult for search methods learn such a relation precisely.

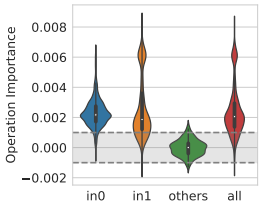
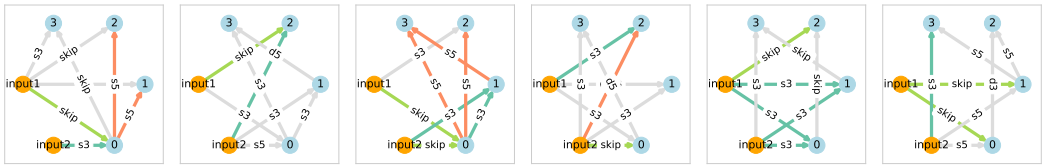


Figure 8: *`skips` are only useful when they form residual links:* `in0` and `in1` denote the residual links formed with either inputs, `others` denote the skip connections not forming residual links and `all` is the overall distribution of OI of skip connections.

The consensus in preferring the aforementioned pattern also extends beyond the training set of NB301: with reference to Fig 9, we select some of the architectures produced by the most recent works that are not represented in the NB301 training set, and it is clear that despite the different search strategies, functional parts of resulting architectures are all characterised by the this pattern of residual connections and separable convolutions, a combination already well-known and well-used



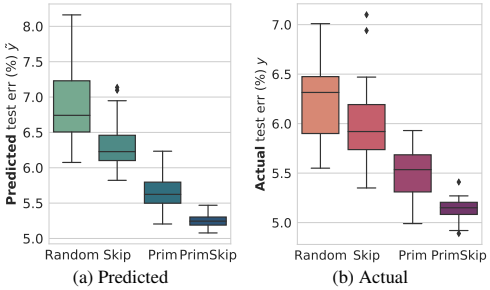
(a) White et al. (2021) (b) Chen et al. (2021b) (c) Li et al. (2021) (d) Ru et al. (2021) (e) Wang et al. (2021c) (f) Chen et al. (2021a)

Figure 9: Normal cells of various SoTA (left to right: BANANAS, DRNAS, GAEA, NAS-BOWL, DARTS_PT and TE-NAS) architectures with the important operations highlighted (the connections to output are omitted since they are all identical across the DARTS search space). Note all cases considered are consistent with the residual link + separable convolution patterns identified, even though the cells and search methods are very different and except for BANANAS, none of the methods here was used to generate the NB301 training set.

both in parts and in sum in successful manually designed networks (e.g. Xception (Chollet, 2017) uses both ingradients). In this sense, many of the existing NAS methods might not have discovered much more novel architectures beyond what we already know; the functional parts of many SoTA NAS architectures could be regarded as variations of the classical architectures, whereas the apparent diversity like the one shown in Fig 2 is often caused by differences in the non-functional parts of the architectures that in fact minimally influence the performances.

***Generating* SoTA architectures** We showed that many NAS architectures share similar traits, but a stronger test is whether these simple patterns alone are *sufficient* for good performance. We construct architectures simply by random sampling, but with 2 constraints, *without additional search*:

1. Normals cell must contain residual link: for architecture generation, we simply manually wire 2 `skips` from both inputs to intermediate node 0 (*Skip* constraint);
2. The other operations are selected from {`s3`, `s5`} only, with all other primitives removed (*Prim* constraint).



(a) Predicted (b) Actual

Figure 10: Distribution of (a) NB301 predicted and (b) actual test error of archs sampled. *Random*: random archs without constraints; *Skip*: archs with residual links and otherwise randomly sampled; *Prim*: random archs using {`s3`, `s5`, `skip`} only. *PrimSkip*: archs satisfying both *Skip* and *Prim*.

While it takes a thorough analysis to study the patterns, the constraints which encode our findings themselves are simple, human-interpretable and moderate (note that only *Skip* is a "hard" constraint specifying exact wiring; *Prim* simply constrains sampling to a smaller subspace). We then sample 50 architectures within both constraints with the same rule for both normal and reduce cells and report their predicted test errors in Fig 10(a). To ensure that we are not biased by the NB301 surrogate, we actually train 30 of the 50 architectures from scratch (protocols specified in App. B.2) and report results in Fig 10(b). To verify the relative importance of each constraint, we also sample the same number of architectures with no constraints or with either constraint activated. We note that both constraints effectively narrow the spread of both predicted and actual test errors, with almost *any* architecture in the *PrimSkip* group performing similarly to the SoTA – only <1% of the training set of NB301 perform better than the mean predicted test error of the *PrimSkip* group (5.24%) in Fig 10(a), while only 5% perform better than the *worst* (5.46%). Apart from the two constraints, the architectures produced are in fact rather varied otherwise in terms of metrics like depth/width and exact wiring (see App. E) – this shows that the two moderate constraints already determine the performance to a great extent, potentially eclipsing other factors previously believed to influence performance. We believe that it might even be possible to fully construct architectures manually from the identified patterns to achieve better results, but the main purpose of this experiment is to show that we may narrowly constrain performance to a very competitive range using very few rules without exactly specifying the cells, instead of aiming for the absolute best architecture in an already noisy search space. Lastly, we analyse NB201 space similarly in App. D and very similar findings hold.

Large architectures We have so far followed the standard NB301 training protocol featuring smalled architectures trained with fewer (100) epochs, generalisation performance on which does not necessarily always transfer to larger architectures (Yang et al., 2020a; Shu et al., 2020). Since the computational cost is much larger, here we first evaluate on 2 randomly selected *PrimSkip* architectures from Sec 4, but stack it into a larger architectures and train longer to make the results comparable those reported the literature. To ensure that the results are *completely* comparable, on CIFAR-10 experiments we do not simply take the baseline results from the original papers; instead,

Table 2: Test error of the state-of-the-art architectures on CIFAR-10 and IMAGENET (mobile setting).

(a) CIFAR-10. All baselines are re-evaluated using the procedure in App. B.2) to ensure the results are completely comparable.

Architecture	Top-1 test error (%) Original	Edited	Edit dist.
DARTSv2 (Liu et al., 2019)	2.44	2.36(−0.08)	1
BANANAS (White et al., 2021)	2.39	2.42(+0.03)	1
DrNAS (Chen et al., 2021b)	**2.27**	2.31(+0.04)	1
GAEA (Li et al., 2021)	2.31	**2.18**(−0.13)	0
NAS-BOWL (Ru et al., 2021)	2.33	2.23(−0.10)	2
NoisyDARTS (Chu et al., 2020)	2.57	2.42(−0.15)	4
DARTS_PT (Wang et al., 2021c)	2.33	2.35(+0.02)	2
SDARTS_PT (Wang et al., 2021c)	2.46	2.36(−0.10)	4
SGAS_PT (Wang et al., 2021c)	2.92	2.48(−0.44)	3
PrimSkip Arch 1	**2.27**	-	-
PrimSkip Arch 2	2.29	-	-

(b) glsImageNet. All baselines are taken from the original papers as re-evaluation is too costly in this case.

Architecture	Test error (%) Top-1	Top-5	Params (M)
DARTSv2 (Liu et al., 2019)	26.7	8.7	4.7
SNAS (Xie et al., 2019b)	27.3	9.2	4.3
GDAS (Dong & Yang, 2020)	26.0	8.5	5.3
DrNAS† (Chen et al., 2021b)	24.2	7.3	5.2
GAEA(C10) (Li et al., 2021)	24.3	7.3	5.3
GAEA(ImageNet)† (Li et al., 2021)	24.0	7.3	5.6
PDARTS (Chen et al., 2019)	24.4	7.4	4.9
PC-DARTS(C10) (Xu et al., 2020)	25.1	7.8	5.3
PC-DARTS(ImageNet)† (Xu et al., 2020)	24.2	7.3	5.3
PrimSkip Arch 1	24.4	7.4	5.7
PrimSkip Arch 2	**23.9**	**7.0**	5.7

†: searched directly on ImageNet.

we obtain the cell specifications provided in some of the most recent papers (see App. E for detailed specifications), re-train each from scratch using a standardised setup (See App. B.3 for details), and show the performance comparison in the "*Original*" column of Table 2a: Recognising that the performance on CIFAR-10 is usually quite noisy (existing papers usually report noise standard deviation of around $0.05 - 0.1\%$), the sampled architectures perform at least on par with the SoTA architectures produced from much more sophisticated search algorithms.

To further verify our findings, we conduct an additional experiment where we edit the SoTA architectures minimally to make them comply to the *PrimSkip* constraints: whenever a cell contains primitives outside {`s3, s5, skip`}, we replace them with ones that are in this set, and we always set the reduce cell to be identical to the normal cell (see App. E for detailed specifications of the architectures). We also create residual links if they are not present, and replace non-residual `skips` with `s3`/`s5` between operations, if any; we do not alter any wiring. Most architectures are already close to conforming to the constraints, so the number of edits required is often small (the edit distance between the editted and original architectures is under "*Edit dist.*" column in Table 2a); in fact, the GAEA cell is already fully-compliant and we only replace its reduce cell with the normal cell). We show the test errors of the edited architectures along with the percentage change from the original ones in "*Edited*" column of Table 2a: the edits result in an improvement in test error up to 0.44% in 6/9 cases, and even where test errors increase after the edits, the differences are very small and probably within margins of error. This shows that at least for the architectures we consider, the SoTA architectures can all be consistently explained by the same simple pattern identified. We finally train the same sampled *PrimSkip* architecture ImageNet using a training protocol strictly comparable to the literature (App. B.3) and the results are shown in Table 2b. Accounting for the estimated evaluation noise on ImageNet[2], it is fair to say that both *PrimSkip* architectures perform on par to, if not better than, the SoTA, even though some of the SoTA architectures are searched on ImageNet directly, an extremely expensive procedure in terms of computational costs.

Discussions We find that the despite the different search strategies and the belief that the search space contains diverse solutions, the key features of many top architectures in the DARTS (and NB201) space are largely similar to each other, and may collectively be viewed as variants to classical architectures. This suggests a gap between the apparent and effective diversity of the search space, potentially explaining the discrepancy between the huge search space and the small performance variability. We also show highly complicated SoTA methods fail to significantly outperform random search with few mild constraints, further demonstrating that it is these simple traits, instead of other complexities, that drives the performance.

As a result, we argue that we should rethink the roles the existing cell-based search spaces play as the key (and sometimes the only) venue on which the search methods develop and iterate. While it is somewhat reassuring we find elements which are known to perform well, this should serve no more than a toy experiment and over-relying on such search spaces could impede progress in NAS: the fact that many algorithms find similar architectures in disguise makes it difficult to conclusively differentiate amongst them. Also, while we do not rule out the possibility that there could be other good-performing architectures not represented by the patterns identified, it is doubtful whether the search space, while nominally diverse, truly contains any novel good-performing architectures beyond what we already know. Consequently, spending vast resources into such a search space seems contradictory to the ambitious goal of NAS to discover novel and diverse architectures *beyond* manual design.

[2]Most NAS papers do not run ImageNet experiments with multiple seeds, but comparable works (Xie et al., 2019a; Goyal et al., 2017) estimate a noise standard deviation of $0.2 - 0.5\%$ in Top-1 error.

5 Related Works

There are multiple previous works that also aim to explain and/or find patterns in cell-based NAS: Shu et al. (2020) find search methods in the DARTS space to favour shallow and wide cells but conclude they do not necessarily generalise better, and hence the pattern does not *explain* performances. Ru et al. (2021) also use subgraphs to explain performances, but only consider first-order Weisfeiler-Lehman (WL) features which could be overly restrictive (note that most subgraphs we find in Fig 7 are not limited to 1-WL features) and specific to the search method proposed. Zela et al. (2020) account for failure mode of differentiable NAS, but ultimately focus on a family of related search methods while the current work is search method-agnostic. On a search space level, a closely related work is Yang et al. (2020a), findings from whom we use extensively, but they mainly identify problems, not explanations; Xie et al. (2019a); Ru et al. (2020) and You et al. (2020) relate performances with graph theoric properties of networks, but it is unclear to what extent do these apply to standard cell-based NAS as the search spaces considered are significantly different (e.g. they typically feature much fewer primitive choices). Lastly, in constructing NAS benchmarks, Dong & Yang (2020); Siems et al. (2020); Ying et al. (2019a) have also provided various insights and patterns, but current work advances such understanding further via a comprehensive and experimentally validated investigation.

6 Suggestions for Future NAS Practices

We believe this work to be useful as an investigation of the existing cell-based NAS as well as to inspire future ones, not only on conventional CNNs but also emerging architectures like transformers. On a search space level, we find a mismatch between the nominal and the effective complexity: complexities are as useful as they contribute to performance and novelty, and thus in a hypothetical new space, we should aim to be aware of these non-functional complexities, and not simply augment the number of primitives available and/or expanding the sizes of the cells. However, identifying such redundancies in a new search space is very challenging a-priori, but fortunately the analysis tools used in this paper are model-agnostic, and thus could be applied to any new search space candidates. Also, while we use the NB301 predictors which train a huge number of architectures to ensure the findings are as representative as possible, we show in App. F that combined with an appropriate surrogate regression model, we may reproduce most findings by training as few as 200 architectures (or 0.4% of the full training set); this suggests that the techniques used could also be cost-effective tools to inspect new search spaces. Another under-explored possibility would be iterative search-and-prune at the search space level, as opposed to the architecture level which is relatively well studied. Using the tools and metrics we introduced to incrementally *grow* the search space from simpler structures and *prune* out those redundant ones in a principled manner.

We believe that a possible reason for many of the current problems is the over-engineered cells and the under-engineered macro-connections. While the cells are more complex than necessary, the connections between them are currently manually fixed in a way that is heavily borrowed from the classical networks. For instance, the manually inserted pooling layers *between* cells might have rendered pooling *within* cells redundant or even harmful, and by adopting macro structures inspired by the manually designed networks, we might also be creating biases that implicitly encourage search methods to discover patterns similar to them. A possible way forward could therefore be simplifying the cells but incorporating variations of connections between the different cells in the search space. The fact that performances depend on a small number of operations suggests that cells may be simplified without sacrificing performance and expressiveness. With a simplified cell, more attention could be shifted to searching for the connections between cells. Such a search space could offer a more expressive range of possible architectures and potentially allow truly novel architectures and/or patterns to be discovered independently from manual designed networks.

7 Conclusion

We present a post-hoc analysis of architectures in the most popular cell-based search spaces. We find a mismatch between the huge nominal complexity and the effective diversity, as many good-performing architectures, despite discovered by very different search methods, share similar traits. We also find many of the design options almost universally adopted to be redundant, as performances of the architectures disproportionately depend on certain operation and connection patterns while the rest are often irrelevant. We conclude that like the rapidly iterating search methods, the search spaces also need to evolve to match the progress of NAS and we provide suggestions based on the main findings in the paper, the latter of which also form some of the most evident directions of future work.

REFERENCES

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *arXiv preprint arXiv:2102.11535*, 2021a.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. *International Conference on Learning Representations (ICLR)*, 2021b.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1294–1303, 2019.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

Xiangxiang Chu, Bo Zhang, and Xudong Li. Noisy differentiable architecture search. *arXiv preprint arXiv:2005.03566*, 2020.

Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. {DARTS}-: Robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KLH36ELmwIB.

Tom Den Ottelander, Arkadiy Dushatskiy, Marco Virgolin, and Peter AN Bosman. Local search is a remarkably strong baseline for neural architecture search. In *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 465–479. Springer, 2021.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *International Conference on Learning Representations (ICLR)*, 2020.

Łukasz Dudziak, Thomas Chau, Mohamed S Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D Lane. Brp-nas: Prediction-based nas using gcns. *Advances in Neural Information Processing Systems (NeurIPS) 33*, 2020.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *J. Mach. Learn. Res.*, 20(177):1–81, 2019.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Andrew Hundt, Varun Jain, and Gregory D Hager. sharpdarts: Faster and more accurate differentiable architecture search. *arXiv preprint arXiv:1903.09900*, 2019.

Chuntao Jiang, Frans Coenen, and Michele Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(1):75–105, 2013.

Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=rkQuFUmUOg3.

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. *International Conference on Learning Representations (ICLR)*, 2021.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *International Conference on Learning Representations (ICLR)*, 2019.

Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *arXiv preprint arXiv:2002.10389*, 2020.

Christoph Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.

Niv Nayman, Yonathan Aflalo, Asaf Noy, and Lihi Zelnik-Manor. Hardcore-nas: Hard constrained differentiable neural architecture search. *International Conference on Machine Learning*, 2021.

Vu Nguyen, Tam Le, Makoto Yamada, and Michael A Osborne. Optimal transport kernels for sequential and parallel neural architecture search. In *International Conference on Machine Learning*, pp. 8084–8095. PMLR, 2021.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pp. 4095–4104. PMLR, 2018.

Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

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, pp. 4780–4789, 2019.

Binxin Ru, Pedro Esperanca, and Fabio Carlucci. Neural architecture generator optimization. *Advances in Neural Information Processing Systems (NeurIPS) 33*, 2020.

Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. *International Conference on Learning Representations (ICLR)*, 2021.

Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. *Advances in Neural Information Processing Systems*, 2020.

Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. *International Conference on Learning Representations (ICLR)*, 2020.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

Xiu Su, Shan You, Tao Huang, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Locally free weight sharing for network width search. *International Conference on Learning Representations*, 2021a.

Xiu Su, Shan You, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. K-shot nas: Learnable weight-sharing for nas with k-shot supernets. *International Conference on Machine Learning*, 2021b.

Dilin Wang, Chengyue Gong, Meng Li, Qiang Liu, and Vikas Chandra. Alphanet: Improved training of supernet with alpha-divergence. *International Conference on Machine Learning*, 2021a.

Jiaxing Wang, Haoli Bai, Jiaxiang Wu, Xupeng Shi, Junzhou Huang, Irwin King, Michael Lyu, and Jian Cheng. Revisiting parameter sharing for automatic neural channel number search. *Advances in Neural Information Processing Systems*, 33, 2020.

Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning action space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021b.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. *International Conference on Learning Representations (ICLR)*, 2021c.

Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. *Advances in Neural Information Processing Systems*, 2020a.

Colin White, Sam Nolen, and Yash Savani. Local search is state of the art for nas benchmarks. *arXiv preprint arXiv:2005.02960*, 2020b.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *AAAI Conference on Artificial Intelligence*, 1(2), 2021.

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1284–1293, 2019a.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *International Conference on Learning Representations (ICLR)*, 2019b.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. *International Conference on Learning Representations (ICLR)*, 2020.

Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? *Advances in Neural Information Processing Systems*, 33, 2020.

Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pp. 721–724. IEEE, 2002.

Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. *International Conference on Learning Representations (ICLR)*, 2020a.

Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *Advances in Neural Information Processing Systems (NeurIPS) 33*, 2020b.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114. PMLR, 2019a.

Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnn explainer: A tool for post-hoc explanation of graph neural networks. *arXiv preprint arXiv:1903.03894*, 2019b.

Jiaxuan You, Jure Leskovec, Kaiming He, and Saining Xie. Graph structure of neural networks. In *International Conference on Machine Learning*, pp. 10881–10891. PMLR, 2020.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *International Conference on Learning Representations (ICLR)*, 2020.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Zongyuan Ge, and Steven Su. Differentiable neural architecture search in equivalent space with exploration enhancement. *Advances in Neural Information Processing Systems*, 33, 2020.

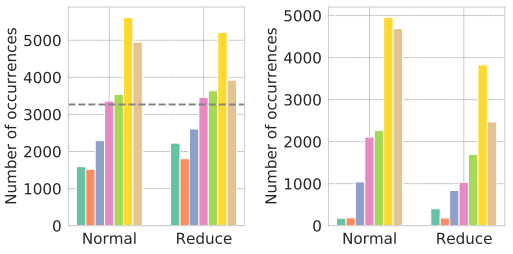
Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *International Conference on Machine Learning*, pp. 12707–12718. PMLR, 2021.

Pan Zhou, Caiming Xiong, Richard Socher, and Steven C. H. Hoi. Theory-inspired path-regularized differential network architecture search. *Advances in Neural Information Processing Systems*, 2020.

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

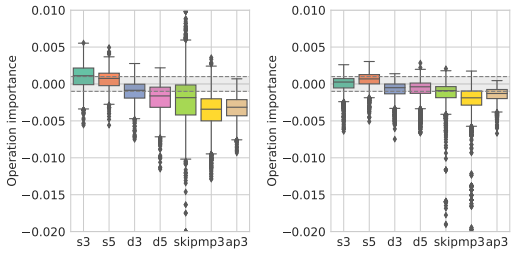
Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A Analysis on Worst-performing Architectures



(a) *All* operations (b) *Important* operations

Figure 11: Distribution of (a) all and (b) important operations by the primitive type of the worst-performing cells. The gray dashed line in (a) denotes the expected number of occurrences if the operations are uniformly sampled in each cell.



(a) Normal cells (b) Reduce cells

Figure 12: Box-and-whisker plots showing the distribution of the operations importance in (a) normal and (b) reduce cells by primitive types. The important operations by the definition of the paper are shown outside the gray shaded area.

We also conduct operation-level analysis on the worst 5% performing architectures of the NB301 training data and the results are shown in Figs 11 and 12, and we find that both pooling operators almost never contribute to good performing architectures as shown in Sec 3, but they actively hurt performance in the poor architectures: from Fig 11, it is clear that the worst performing cells are both characterised by large number of pooling operators (Fig 11(a) and a large number of *important* pooling operators actively degrading performance: this is unsurprising and also pointed out in the analysis in the original NB301 paper (Siems et al., 2020) as cells with a large number of pooling operations aggressively cause loss of information. Other than that, both dilated convolution operations remain rather neutral and the separable convolutions remain positive in operation importance even in poorly-performing architectures. Skip connections in this case are quite negative in general but still have a very large spread – this could be either due to a large number of skip connections in the cell which is a known failure mode of many differentiable NAS algorithms Zela et al. (2020) or that skip connections need to be paired with separable convolutions as shown in Fig 7 for positive effects, which is not the case in these poor architectures where separable convolutions are underrepresented.

We also repeat the subgraph-level analysis on these architectures (Fig 13). An interesting insight is that the poorly performing subgraphs are much more "diverse" than the good ones, and the primitives in the two groups almost never overlap: the none of positive subgraphs in 7 contains `{d3, d5, mp3, ap3}`, whereas none of the negative subgraphs contains `{s3, s5}`. This shows that in the present search space the primitives are somewhat "separable" and the redundant primitives `{d3, d5, mp3, ap3}` may simply be discarded without affecting the resulting performances – we argue this should not be the case in a well-designed ideal search space. In principle, every primitive should be a building block that that potentially contribute to architectures positively at least in some cases.

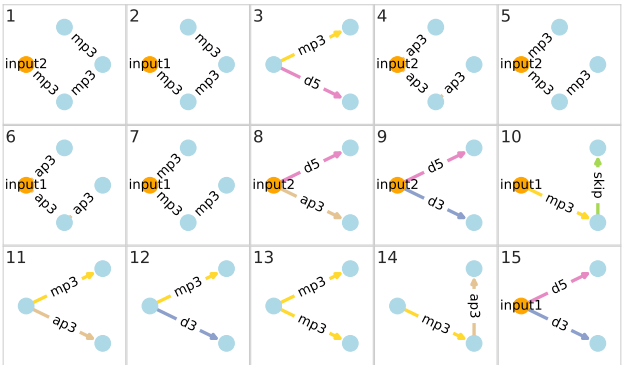


Figure 13: Frequent subgraphs in the good-performing architectures ranked by ratio of supports between the important subgraphs and the reference and properties of the discovered frequent subgraphs in the worst-performing architectures.

B IMPLEMENTATION DETAILS

B.1 DATA

We primarily use the data from the training set of the NB301 benchmark (Siems et al., 2020), available at the official repository at https://github.com/automl/nasbench301. To obtain a sound performance surrogate over the entire DARTS search space, NB301 trains more than 50,000 architectures using the protocol listed in App B.2, and use the architectures and their corresponding test performance on CIFAR-10 as inputs and labels to train a number of surrogate models as performance predictors, including GIN, XGBoost and LGBoost (in this paper, we always use the XGBoost surrogate as it is shown to be the best on balance according to Siems et al. (2020)). The architectures are produced from a number of technically diverse methods representing almost all mainstream genres of cell-based NAS like gradient-based methods, Bayesian optimisation, reinforcement learning and simpler methods like local search and random search: DARTS (Liu et al., 2019), DRNAS (Chen et al., 2021b), GDAS (Dong & Yang, 2020), reinforcement learning (RL) (Zoph & Le, 2016), differential evolution (DE) (Price et al., 2006), PC_DARTS (Xu et al., 2020), Tree parzen estimator (TPE) (Bergstra et al., 2011), local search (LS) (Den Ottelander et al., 2021; White et al., 2020b) and regularised evolution (RE) (Real et al., 2019).

B.2 TRAINING PROTOCOLS ON DARTS ARCHITECTURES

We exactly follow the NB301 protocols for all experiments involving architecture training (except for the larger architectures on CIFAR-10 and ImageNet, which we outline below at App B.3). Specifically, we train architectures obtained from stacking the cells 8 times (8-layer architectures) with initial channel count of 32 on the CIFAR-10 dataset using the standard train/val split, and we use the hyperparameters below on a single NVIDIA Tesla V100 GPU:

```
Optimizer: SGD
Initial learning rate: 0.025
Final learning rate: 1e-8
Learning rate schedule: cosine annealing
Epochs: 100
Weight decay: 3e-4
Momentum: 0.9
Auxiliary tower: True
Auxliary weight: 0.4
Cutout: True
Cutout length: 16
Drop path probability: 0.2
Gradient clip: 5
Batch size: 96
Mixup: True
Mixup alpha: 0.2
```

B.3 EVALUATION PROTOCOLS ON DARTS ARCHITECTURES

CIFAR-10 For the evaluation, we use larger architectures obtained from stacking the cells 20 times (20-layer architectures) with an initial channel count of 36 on the CIFAR-10 dataset. The other hyperparameters (mostly consistent with those used in App B.2 except for the number of epochs trained) used are:

```
Optimizer: SGD
Initial learning rate: 0.025
Final learning rate: 1e-8
Learning rate schedule: cosine annealing
Epochs: 600
Weight decay: 3e-4
Momentum: 0.9
Auxiliary tower: True
Auxliary weight: 0.4
Cutout: True
Cutout length: 16
```

```
Drop path probability: 0.2
Gradient clip: 5
Batch size: 96
Mixup: True
Mixup alpha: 0.2
```

This protocol is identical to the original DARTS protocol (Liu et al., 2019), with the only exception that to be consistent with the NB301 protocol, we also incorporate the Mixup regularisation (Zhang et al., 2017) during evaluation. This accounts for the fact that the accuracy reported in this paper is generally better than those reported in the literature. However, as mentioned in the main text, we re-train every architectures from the scratch, including the baselines, using the identical protocol listed above, instead of simply taking the numbers from the original papers. As a result, no architecture has been given unfair advantage because of the more effective regularisation used in this paper. We also conduct all experiments on a single NVIDIA Tesla V100 GPU.

ImageNet On ImageNet, we use a protocol that is identical to Chen et al. (2021b). It is also almost identical to those used in Xu et al. (2020); Chen et al. (2019); Liu et al. (2019) except for batch sizes (which depend on the availability of hardware; larger batch size is only available for a parallel many-GPU setup) and the corresponding linear scaling in learning rates. Specifically, we form 14-layer architectures with an initial channel count of 48 using 8$\times$ NVIDIA Tesla V100 GPUs. Note that since we are unable to re-evaluate all the baselines using a standardised training protocols in this case due to the extreme computational cost, we use a protocol that strictly adheres to the existing works with the additional Mixup regularisation in CIFAR-10 disabled in the ImageNet experiments to ensure the comparability of the results. The other hyperparameters are as followed:

```
Optimizer: SGD
Initial learning rate: 0.5
Learning rate schedule: linear annealing
Epochs: 250
Weight decay: 3e-5
Momentum: 0.9
Auxiliary Tower: True
Auxliary weight: 0.4
Cutout: True
Cutout length: 16
Drop path probability: 0
Gradient clip: 5
Label smoothing: 0.1
Mixup: False
Batch size: 768
```

C List of Referenced Papers

We present the details covered in our preliminary survey on the NAS search methods papers published in top machine learning conferences during the past year in Table 3.

Table 3: A list of NAS methods papers (i.e. excluding, e.g. review or benchmark papers) published in the past year in top machine learning conferences. *Cells-based* means the work demonstrates at least one part of the major results in the DARTS cell-based search space and/or highly related ones (such as the various NAS-Benches and/or those otherwise highly resemble DARTS). *Cells-only* means the works *only* demonstrate the results in aforementioned search space(s). Whenever a paper is not cell-based or cells-only, *other spaces evaluated* shows the alternative spaces the papers report results on. The list is potentially incomplete, as we only select papers that explicitly mention NAS in the title and/or the abstract.

Venue	Name	Reference	Cells-based	Cells-only	Other spaces evaluated
NeurIPS 2020	BRP-NAS	Dudziak et al. (2020)	✓	✓	
	NAGO	Ru et al. (2020)			NAGO space
	ISTA-NAS	Yang et al. (2020b)	✓	✓	
	arch2vec	Yan et al. (2020)	✓	✓	
	-	White et al. (2020a)	✓	✓	
	PR-DARTS	Zhou et al. (2020)	✓	✓	
	E^2NAS	Zhang et al. (2020)	✓	✓	
	APS	Wang et al. (2020)			Channel/width search
	SemiNAS	Luo et al. (2020)	✓		MobileNet space
	BONAS	Shi et al. (2020)	✓	✓	
ICLR 2021	NAS-BOWL	Ru et al. (2021)	✓	✓	
	DrNAS	Chen et al. (2021b)	✓	✓	
	GAEA	Li et al. (2021)	✓	✓	
	DARTS-	Chu et al. (2021)	✓	✓	
	TE-NAS	Chen et al. (2021a)	✓	✓	
	DARTS_PT, etc	Wang et al. (2021c)	✓		MobileNet space
	MetaD2A	Lee et al. (2021)	✓		MobileNet space
	CafeNet	Su et al. (2021a)			Channel/width search
ICML 2021	BO-TW/kDPP	Nguyen et al. (2021)	✓	✓	
	AlphaNet	Wang et al. (2021a)			MobileNet space
	CATE	Wang et al. (2021a)	✓	✓	
	HardCoRe-NAS	Nayman et al. (2021)			MobileNet space
	K-Shot NAS	Su et al. (2021b)	✓		MobileNet space
	Few-shot NAS	Zhao et al. (2021)	✓		ProxylessNAS space, RNN, AutoGAN
Total	**24**		**19** (79%)	**14** (58 %)	

D ANALYSIS ON NAS-BENCH-201

Fig 14 shows the NB201 search space, a popular NAS benchmark commonly used that is highly similar to the DARTS cell, but 1) only one cell (instead of two) is searched, 2) each cell is connected to its immediate preceding layer only, and 3) is with a reduced set of primitives. Also, unlike the DARTS cell, all edges in the NB201 cell are enabled.

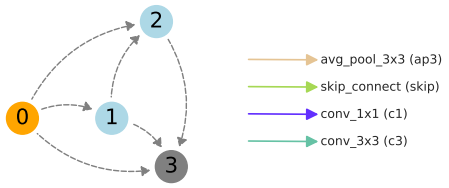


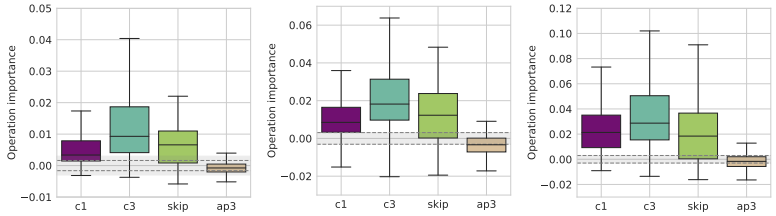
Figure 14: The NB201 (Dong & Yang, 2020) cell, which is highly similar to the DARTS space but much simpler. All 6 locations (denoted by gray dashed arrows) are available for search, and each is filled by one out of the four candidate primitives (or `None`, which disables the edge).

We also conduct a brief analysis in a similar manner to the main text on top 5% performing architectures on NB201 dataset, and we show the operation importance distribution of each primitive in Fig 15. We observe that due to the smaller cell size and the primitive set, the operations in a NB201 cell is typically more important and the only redundant operations is `ap3`. We hypothesise that the reason is similar to the DARTS search space as the manually specified macro connection between the cells already include pooling operations, rendering them unncessary within the cells.

The second experiment to conduct is verifying whether in the NB201 search space the good performing cells are also characterised by the patterns we identified in Sec 4. To do so, we adapt the *Skip* and *Prim* constraints in the NB201 space:

1. *Skip* constraint: in the NB201 search space, the only way to form a residual connection is to place `skip` on edge $0 \rightarrow 3$ (with reference to Fig 14.
2. *Prim* constraint: apart from the manually specified edge, all other operations are sampled from the reduced primitive set `{c1, c3}` consisting of convolutions only.

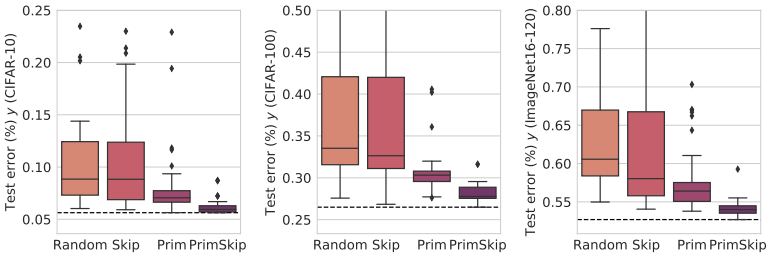
Similar to our procedure in Sec 4, we sample 50 architectures within each group (no constraint, either constraint and both constraints), and we show their test performance in Fig 16. It is also worth noting



(a) CIFAR-10 (b) CIFAR100 (c) ImageNet16-120

Figure 15: Box-and-whisker plots showing the distribution of the operation distribution in NB201 benchmark on (a) CIFAR-10, (b) CIFAR-100 and (c) ImageNet16-120 datasets. The gray shaded areas denote the noise standard deviation which differs in each dataset.

that the ground-truth optimum in each dataset is known in NB201 and is accordingly marked in Fig 16. Differing from the observations in DARTS search space results, in this case *Skip* constraint alone does not impact the performance significantly, but again the *PrimSkip* group with both constraints activated perform in a range very close to the optimum: in fact, the optimal architectures in all 3 datasets, while different from each other, all belong to the *PrimSkip* group and are found by random sampling with fewer than 50 samples. This again confirms that our findings in the main text similarly generalise to NB201 space.



(a) CIFAR-10 (b) CIFAR100 (c) ImageNet16-120

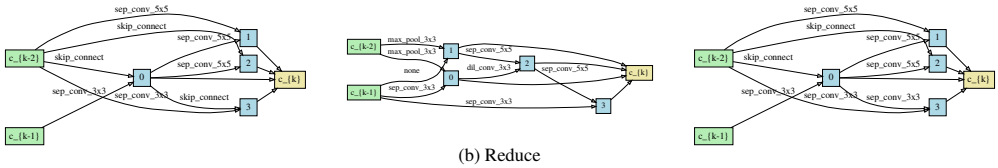
Figure 16: Distribution of the test errors on (a) CIFAR-10, (b) CIFAR-100 and (c) ImageNet of NB201 architectures. Note that since NB201 is a tabular benchmark that exhaustively trains and evaluates all the architectures within its search space, all test errors reported here are actual, not predicted.

E ARCHITECTURE SPECIFICATIONS

In this section, we show the specifications of (a.k.a genotypes) the different architectures in the DARTS search space.

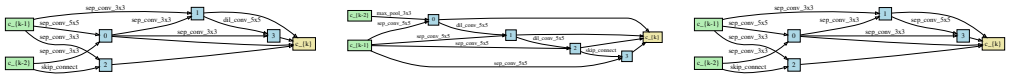
E.1 ORIGINAL AND EDITED GENOTYPES FROM BASELINE PAPERS

Here we show the genotypes original and edited (corresponding to the results in the *"Edited"* column in Table 2a) architectures (Fig. 17 – 24). In all figures, "Normal" and "Reduce" denote the normal and reduce cells of the *Original* architectures where "Edited" denote the normal and reduce cells of the edited architectures (note that the edited architectures always have identical normal and reduce cells).



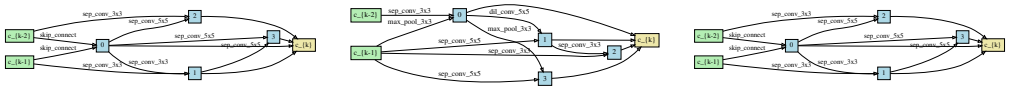
(a) Normal (b) Reduce (c) Edited

Figure 17: Genotypes of BANANAS architecture (White et al., 2021)



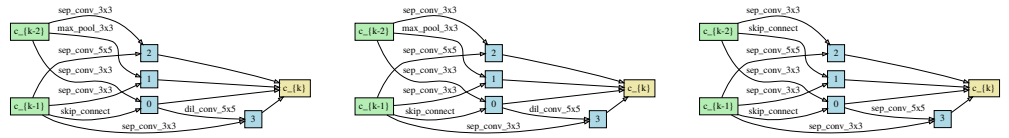
(a) Normal (b) Reduce (c) Edited

Figure 18: Genotypes of DRNAS architecture (Chen et al., 2021b)



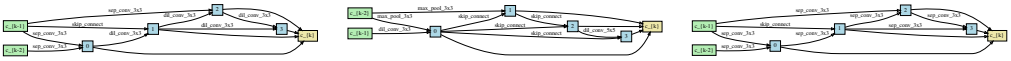
(a) Normal (b) Reduce (c) Edited

Figure 19: Genotypes of GAEA architecture (Li et al., 2021). Note that the edited genotype is identical to the original normal genotype as it is already compliant with both *Prim* and *Skip* constraints.



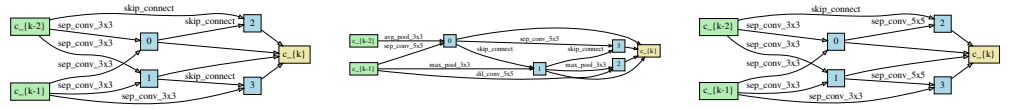
(a) Normal (b) Reduce (c) Edited

Figure 20: Genotypes of NASBOWL architecture (Ru et al., 2021).



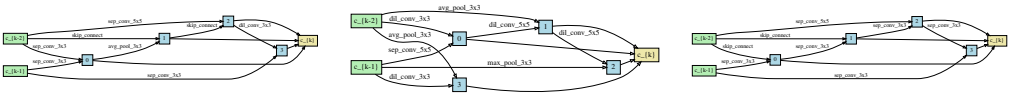
(a) Normal (b) Reduce (c) Edited

Figure 21: Genotypes of NOISYDARTS architecture (Chu et al., 2020)



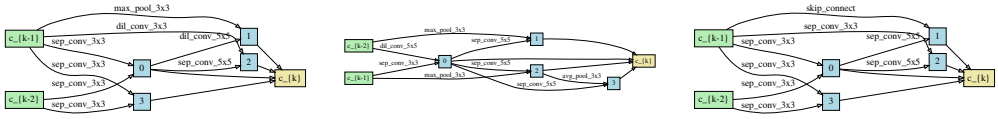
(a) Normal (b) Reduce (c) Edited

Figure 22: Genotypes of DARTS_PT architecture (Wang et al., 2021c)



(a) Normal (b) Reduce (c) Edited

Figure 23: Genotypes of SDARTS_PT architecture (Wang et al., 2021c)



(a) Normal (b) Reduce (c) Edited

Figure 24: Genotypes of SGAS_PT architecture (Wang et al., 2021c)

E.2 Random Genotypes sampled in the PrimSkip Group

We show some examples of the genotypes generated via the constrained random sampling in the *PrimSkip* group in Sec 4 in Fig 25, while the two architectures selected for the CIFAR-10/ImageNet experiments on the larger architectures is shown in Figs 26 and 27.

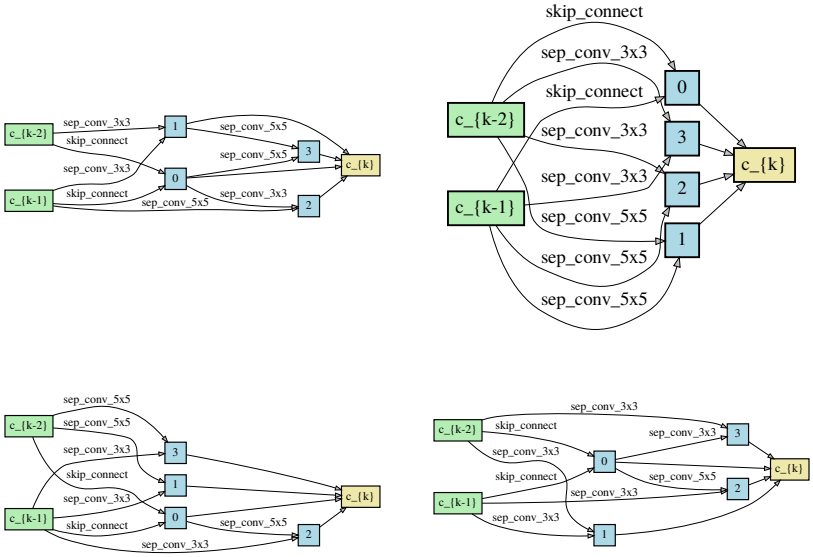
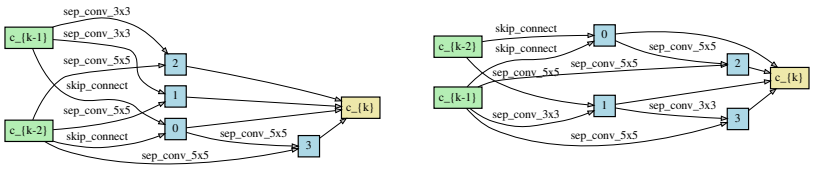
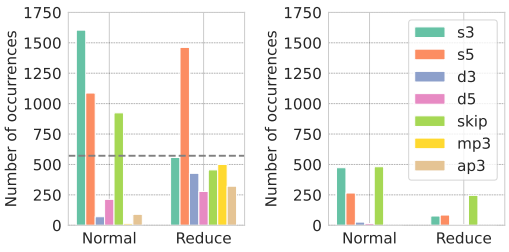


Figure 25: Some of the randomly sampled architectures in the *PrimSkip* group.



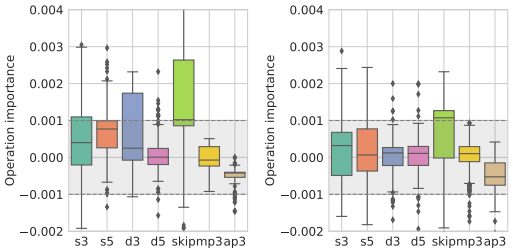
(a) Normal (b) Reduce

Figure 26: Randomly selected *PrimSkip* architecture 1 for the experiments on the larger architectures



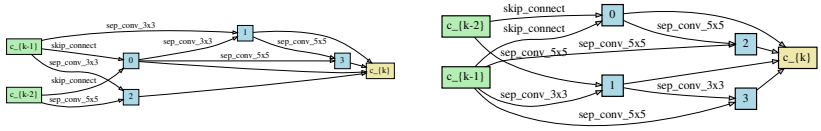
(a) *All* operations (b) *Important* operations

Figure 28: Distribution of (a) all and (b) important operations by the primitive types *according to the* BANANAS *surrogate*. The gray dashed line in (a) denotes the expected number of occurrences if the operations are uniformly sampled.



(a) Normal cells (b) Reduce cells

Figure 29: Box plots showing the distribution of the operations importance in (a) normal and (b) reduce cells *according to the* BANANAS *surrogate*. The important operations by the definition of the paper are shown outside the gray shaded area.



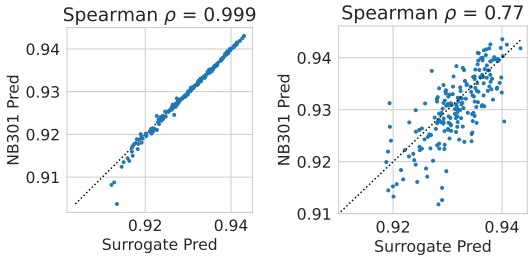
(a) Normal (b) Reduce

Figure 27: Randomly selected *PrimSkip* architecture 2 for the experiments on the larger architectures

F REPRODUCING RESULTS WITH LESS TRAINING DATA

In this section, we show that it is possible to reproduce many results in the paper using less than 0.4% of the data compared to the full set of more than 50,000 architecture-performance pairs used in the NB301 surrogate, thereby motivating the use of the tools introduced in this paper as a generic, cost-effective search space inspector: For the experiments conducted, we use the surrogate from BANANAS (White et al., 2021), which combines a neural ensemble predictor with path encoding of the architectures – it is worth noting that alternative surrogates may also be used, but it is preferable to use a sample-efficient surrogate that is capable of finding meaningful relations in the input data with a modest number of evaluations. Specifically, we randomly sample 200 architectures from the search space and query their NB301 predicted performance as a proxy for the ground-truth performance. We then compute the path encoding of each architecture and train a predictor with the default hyperparameters from White et al. (2021).

We first verify whether the surrogate using less data is able to learn meaningful patterns by drawing another 200 random unseen architectures in the search space and compare the predictions by the neural ensemble predictor vs the NB301 prediction (Fig 30) and it is clear that the regression performance is already satisfactory (with a Spearman rank coefficient of 0.77) despite using much less data.



(a) Train (b) Validation

Figure 30: Regression performance of the BANANAS predictor with 200 training data vs the NB301 surrogate with more than 50,000 training data.

We then repeat the analysis in the main text, and show the operation-level findings (Sec 3) in Figs 28 and 29 and the important subgraphs corresponding to Sec 4 in Fig 31. It is worth noting that most of

the findings are already highly similar to those in the main text, although, for example, the operation importance distributions in Fig 29 have a larger variance due to the less certain predictions. The main purpose of this study is to show that combined with the explanability tools used in the present work, an appropriate performance surrogate, which is only used as a vessel towards searching in some search methods so far, can be itself valuable. We demonstrate that it could shed insights into the strengths and weaknesses of an arbitrary search space with a modest number of observations – for example, during design of a new search space, we may randomly sample and evaluate a modest number of architectures and similarly fit a surrogate. We may then use the explainability tool to inspect the search space in a similar procedure in this section – we believe this could potentially prevent some of the pitfalls described in the existing cell-based search spaces to recur in prospective new ones during the design process.

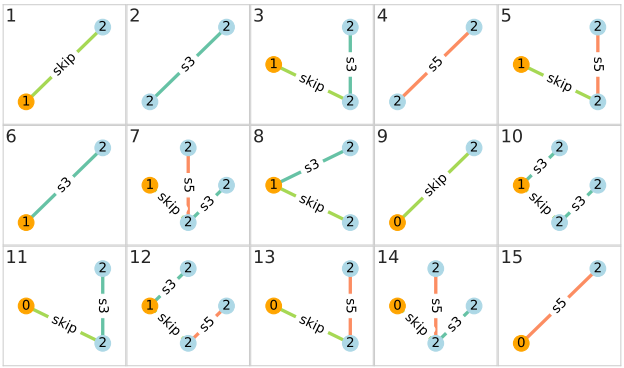


Figure 31: Frequent subgraphs in the good-performing architectures ranked by ratio of supports between the important subgraphs and the reference and properties of the discovered frequent subgraphs *according to the* BANANAS *surrogate*. Note that the residual link + separable convolution patterns are highly similar to those identified in Fig 7 in the main text