Efficient Architecture Search for Diverse Tasks

Anonymous Author(s)
Affiliation
Address
email

Abstract

While neural architecture search (NAS) has enabled automated machine learning (AutoML) for well-researched areas, its application to tasks beyond computer vision is still under-explored. As less-studied domains are precisely those where we expect AutoML to have the greatest impact, in this work we study NAS for efficiently solving *diverse* problems. Seeking an approach that is fast, simple, and broadly applicable, we fix a standard convolutional network (CNN) topology and propose to search for the right kernel sizes and dilations its operations should take on. This dramatically expands the model's capacity to extract features at multiple resolutions for different types of data while only requiring search over the operation space. To overcome the efficiency challenges of naive weight-sharing in this search space, we introduce DASH, a differentiable NAS algorithm that computes the mixture-of-operations using the Fourier diagonalization of convolution, achieving both a better asymptotic complexity and an up-to-10x search time speedup in practice. We evaluate DASH on ten tasks spanning a variety of application domains such as PDE solving, protein folding, and heart disease detection. DASH outperforms state-of-the-art AutoML methods in aggregate, attaining the best-known automated performance on seven tasks. Meanwhile, on six of the ten tasks, the combined search and retraining time is less than 2x slower than simply training a CNN backbone that is far less accurate.

1 Introduction

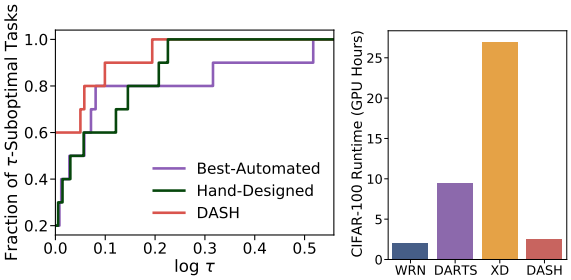
The success of deep learning for computer vision and natural language processing has spurred growing interest in enabling similar breakthroughs for other domains such as biology, healthcare, and physical sciences. Consequently, there is enormous potential for neural architecture search to help automate model development in these diverse areas. However, while extensive NAS research has been devoted to improving the search speed [1, 2] and automatically attaining state-of-the-art performance on vision datasets such as CIFAR and ImageNet [3], the resulting algorithms have subpar performance beyond the tasks on which they were developed. For example, in the analysis of NAS-Bench-360 [4], a recent benchmark designed for improving task diversity in NAS evaluation, the authors showed a significant gap between models found by NAS methods, such as DARTS [2] and DenseNAS [5], and hand-crafted expert architectures on a number of distinct applications.

To improve the generalizability of AutoML methods, recent works such as AutoML-Zero [6] and XD-operations [7] propose to relax the inductive biases encoded in the standard search spaces. However, the former is not designed for practical deployment, and the latter is too expensive to execute even for simple problems like CIFAR-100 (Fig. 1b). Hence we ask: *is there an approach that can provide sufficient expressivity to yield high accuracy across multiple domains, while still retaining the faster search and the more efficient final models of discrete architecture search?*

In this work, we answer in the affirmative by introducing a novel NAS method called DASH (**D**iverse-task **A**rchitecture **S**earc**H**). In order to attain *multi-domain capability*, DASH adapts standard CNN backbones to various learning problems by finding substitutes for their layer operations.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Concretely, we consider a search space of cross-scale dilated convolutions which are effective for multi-scale feature extraction [8, 9] and context aggregation [10, 11]. Our key difference from past search spaces is that we explicitly consider filters with *a wide range of kernel sizes and dilations*—while most NAS methods only handle kernels with maximum size 5 and dilation rate 2, our proposed operator space includes not only the conventional small kernels but also significantly larger ones with size 15 or dilation 127 (Fig. 2). This design choice is motivated by the fact that large kernels can capture input relations for dense prediction problems [12], model long-range dependencies for sequence tasks [13, 14], and resemble global-attention in Transformers [15]. Thus, DASH's cross-scale search space enables adaptation to diverse downstream tasks, unlike prior NAS work which targets image classification and assumes that small kernels are sufficient.



(a) (b)

Figure 1: (a) Comparing the aggregate performance of the best AutoML methods (task-wise), hand-designed models, and DASH on ten diverse tasks via performance profiles (defined in Section 4.1). Larger values (larger fractions of tasks on which a method is within τ-factor of the best) are better. (b) Runtime for Wide ResNet, DARTS, XD, and DASH on CIFAR-100. XD is too expensive to be applied to other tasks considered in this work [4, 7].

However, *efficiently* searching for an appropriate kernel configuration in this expansive cross-scale search space is non-trivial. Indeed, for existing NAS algorithms, the cost of exploring a combinatorially large set of operators is substantial. Even for weight-sharing methods that are known for efficiency, e.g., DARTS [16], the computational complexity scales directly with the number of kernels considered and quadratically with the largest kernel size. To overcome this obstacle, DASH explores multi-scale convolutions via three techniques—the first two exploit mathematical properties of convolutions, and the last one takes advantage of fast matrix multiplication on GPUs. Specifically:

1. Using the **linearity** of convolutions, we mix several convolutions by computing one convolution equipped with a combined kernel rather than applying each filter separately and aggregating multiple outputs. While the number of convolution computations required by the naive aggregation of $|K|$ possible kernel sizes and $|D|$ possible dilations is $\mathcal{O}(|K||D|)$, our approach has $\mathcal{O}(1)$ complexity, independent of the search space size.
2. Using the **diagonalization** of convolutions, we relegate a major portion of the computation to element-wise multiplication in the Fourier domain, minimizing the effect of the largest kernel size on the complexity of our algorithm. For instance, a standard 1D convolution requires $\mathcal{O}(nk)$ operations to convolve a size-k kernel with a length-n input, but a Fourier convolution takes only $\mathcal{O}(n \log n)$, a critical improvement that makes searching over large kernels significantly easier.
3. Our final strategy is to use Kronecker products of undilated kernels and small sparse matrices to compute dilated kernels quickly on GPUs. This brings an additional two-fold speedup on top of the previous techniques.

Aside from these innovations, DASH employs the standard weight-sharing scheme of training a supernet, discretizing to obtain a model, and retraining the model for end tasks [2]. We analyze the asymptotic complexity of the first two techniques and verify the practical utility when all three are combined together. In particular, DASH achieves a ten-fold speedup in total for differentiable NAS over the multi-scale search space. Moreover, we show that searching over large kernels is necessary to solve diverse problems and that each technique on its own cannot scale in this large-kernel setting.

In terms of accuracy performance, we evaluate DASH on ten datasets spanning multiple application areas such as PDE solving, protein folding, and disease prediction from NAS-Bench-360 [4]. As shown in Fig. 1, DASH yields models with better aggregate performance than those returned by leading AutoML methods as well as hand-designed task-specific architectures. As for individual tasks, DASH beats all past automated approaches on seven of the ten problems and exceeds hand-designed models on six, simultaneously maintaining strong efficiency relative to weight-sharing methods like DARTS. The empirical success of DASH implies that CNNs with appropriate kernels can be competitive baselines for problems where expert architectures are not available.

2 Related Work

Neural architecture search (NAS) aims to automate the design of neural networks. Recently, there has been significant progress in both search space design [2, 7, 17] and search strategy development [1, 2, 3, 18, 19]. However, these methods are mostly evaluated on image classification or segmentation, with a few focusing on new applications such as image restoration [20], audio classification [21], and machine translation [22]. Problems beyond the vision and language domains are less-explored.

In this work, we seek to improve the generalizability of NAS by a morphism-based approach [7, 23, 24] that adapts existing CNN backbones to target tasks. We specifically focus on expanding the operation space to multiple types of convolutions, simultaneously varying the kernel size and the dilation factor. Past work in related directions has at most studied the easier problem of altering dilation alone, and only for vision tasks [25]. Therefore, although convolution has been an integral part of NAS, how to search over a large set of convolutional operators remains an open problem. In the following, we identify four types of solutions from existing work and illustrate their limitations.

Differentiable Architecture Search (DARTS) We can treat convolutions as ordinary operators and apply a scalable NAS algorithm. In particular, DARTS [2] introduces continuous relaxation to the weight-sharing paradigm [1] and allows us to gain information about many networks efficiently by training a combined supernet. The algorithm relaxes the discrete set of operations at each edge in a computational graph as a softmax so the search process is end-to-end differentiable and amenable to regular optimizers. After search, it discretizes the weights to output a valid architecture. The original DARTS search space contains only four convolutions with kernel sizes no larger than 5 and dilation rates no larger than 2. While this small search space might be enough for low-resolution image input, it is insufficient for diverse tasks such as high-dimensional time series problems [4]. Although one could add more convolutions one-by-one to the operator set to augment performance, this approach scales poorly, as reflected in the limited search spaces of similar methods like AMBER [26]. In fact, AMBER has to shift the kernel size up to achieve good performance on long-sequence genomic data.

MergeNAS and RepVGG An alternative way to explore the convolutional search space is to take advantage of the operator's linearity. That is, we can first mix the kernels and then apply convolution once, unlike DARTS which computes each convolution separately and outputs the aggregated result to the next layer. This kernel-mixing strategy, which we call *mixed-weights* and will formally define in Section 3.2, has been employed by MergeNAS [27] and RepVGG [28] to improve architecture search robustness and VGG inference speed, respectively. It works well for *a few small* kernels. However, we will show later that similar to DARTS, *mixed-weights* on its own is also insufficient for searching over *a diverse set of large kernels* which is crucial to solving a wide range of problems.

Expressive Diagonalization (XD) Apart from linearity, XD-operations [7] propose to utilize the convolutional theorem in architecture search. In particular, XD expresses the convolution acting on input $\mathbf{x}$ with filter $\mathbf{w}$ as $\mathbf{K}\,\mathrm{diag}(\mathbf{L}\mathbf{w})\mathbf{M}\mathbf{x}$, where $\mathbf{K}, \mathbf{L}, \mathbf{M}$ are appropriate discrete Fourier transforms, and constructs an expansive search space by replacing these transforms with searched kaleidoscope matrices [29]. Although this new search space includes all types of convolutions, the search process is unacceptably long even for simple benchmarking tasks such as CIFAR-100 (Fig. 1b), let alone the more complex set of diverse problems that we consider in this paper. In addition, the output architectures of XD are as inefficient as the supernet due to the absence of a discretization step.

Single-Path NAS Lastly, Single-Path NAS [30] defines a large filter and uses its subsets for smaller filters. This DARTS-based method compensates operator heterogeneity for efficiency during search. It does not handle search spaces with many large kernels and is not evaluated on diverse tasks.

Outside the field of AutoML, there is also emerging interest in designing general-purpose models such as Perceiver IO [31] and Frozen Pretrained Transformer [32]. However, these Transformer-based models do not adapt the network to the target tasks and are generally harder to train compared with CNNs. In Table 2, we evaluate Perceiver IO and show that its performance is not ideal.

3 Methods

Now, we describe the details of DASH (Algorithm 1). We first explain how DASH leverages existing networks to initialize the supernet and generate different models for diverse tasks. Then, we formally define the multi-scale convolution search space and propose a fast way to search this space using the three efficiency-motivated techniques mentioned earlier. Finally, we outline the procedure for discretizing the search space and retraining the searched model.

Algorithm 1 DASH

Input: training data Z, loss function l, the set of kernel sizes K, the set of dilation rates D, and subsampling ratio p
Initialize the backbone and replace each $\mathbf{Conv}$ layer with the mixed operation $\mathbf{AggConv}_{K,D}$
while not converged **do**
  Subsample $p|Z|$ training points uniformly at random
  Compute forward pass using Equation 4
  Descend the architecture parameters α by $\nabla_\alpha l(\mathbf{w}, \alpha)$ and the model weights $\mathbf{w}$ by $\nabla_{\mathbf{w}} l(\mathbf{w}, \alpha)$
end while
Select $\arg\max_{k \in K, d \in D} \alpha_{k,d}$ for each $\mathbf{AggConv}$ layer
Tune retraining hyperparameters on a validation subset of the training data
Retrain the discretized model with all training data

3.1 Decoupling Topology and Operations

Every architecture is a mapping from model weights to functions and can be described by a directed acyclic graph $G(V, E)$. Each edge in E is characterized by $(u, v, \mathbf{Op})$, where $u, v \in V$ are nodes and $\mathbf{Op}$ is an operation applied to u. Node v aggregates the outputs of its incoming edges. NAS aims to automatically select the edge operations and the graph topology to optimize some objective. For each edge, $\mathbf{Op}$ is chosen from a search space $S = \{\mathbf{Op}_a \,|a \in A\}$ where $a \in A$ are architecture parameters. In past work, A usually indexes a small set of operations. For instance, the DARTS search space specifies $A_{discrete} = \{1, \ldots, 8\}$ with $S = \{\mathbf{Zero}, \mathbf{Id}, \mathbf{MaxPool}_{3\times3}, \mathbf{AvgPool}_{3\times3}, \mathbf{Conv}_{3\times3}, \mathbf{Conv}_{5\times5}, \mathbf{DilatedConv}_{3\times3,2}, \mathbf{DilatedConv}_{5\times5,2}\}$. However, A can also be defined systematically to identify operator properties, e.g., $\{(\text{kernel size } k, \text{dilation } d)\}$ for convolutions.

A common way to determine the network topology is to search for blocks of operations and stack several blocks together. In this work, we take a different, morphism-based approach [7, 23, 24]: we use existing networks as backbones and replace certain layers in the backbone with the searched operations. Specifically, we select a set of architectures to accommodate both 2D and 1D datasets. Convolutional layers with different kernels can then be plugged into these networks. An advantage of decoupling topology and operation search is flexibility: the searched operators can vary from the beginning to the end of a network, so features at different granularities can be processed differently.

We pick Wide ResNets (WRNs) [33, 34] as the backbone networks due to their simplicity and effectiveness in image and sequence modeling. Before search, the supernet is initialized to the backbone. Then, all $\mathbf{Conv}$ layers are substituted with an operator $\mathbf{AggConv}_{K,D}$ (short for aggregated convolution) that represents the new search space which we now define. For simplicity, our mathematical discussion will stick to the 1D case, though our experiments are on both 1D and 2D data.

3.2 Efficiently Searching for Multi-Scale Convolutions

A convolution filter is specified by the kernel size k and the dilation rate d (we do not consider stride which does not change the filter shape). The effective filter size is $(k-1)d+1$ with nonzero entries separated by $d-1$ zeros. Let $\mathbf{Conv}_{k,d}$ be the convolution with kernel size k, dilation rate d, c_{in} input channels, and c_{out} output channels. Given input data with shape n, let K be our interested set of kernel sizes, D the set of dilations. We define the $\mathbf{AggConv}_{K,D}$ search space as

$$S_{\mathbf{AggConv}_{K,D}} = \{\mathbf{Conv}_{k,d} \,|k \in K, d \in D\}. \tag{1}$$

Hence, $A = K \times D$ in previous notations. $S_{\mathbf{AggConv}_{K,D}}$ contains a collection of convolutions with receptive field size ranging from k_{min} to $d_{\max}(k_{\max} - 1) + 1$, which allows us to discover models that process the input with varying resolution at each layer.

To retain the efficiency of discrete NAS, we apply the continuous relaxation scheme of DARTS to $S_{\mathbf{AggConv}_{K,D}}$, which mixes all operations in the space using architecture parameters $\{\alpha_{k,d} \in \triangle_{|K||D|} | (k,d) \in K \times D\}$[1], so the output of each edge in the computational graph is

$$\mathbf{AggConv}_{K,D}(\mathbf{x}) := \sum_{k \in K} \sum_{d \in D} \alpha_{k,d} \cdot \mathbf{Conv}(\mathbf{w}_{k,d})(\mathbf{x}). \tag{2}$$

[1] $\triangle$ denotes the probability simplex.

Table 1: Complexity of different methods for computing $\mathbf{AggConv}$. For notation, $\bar{K} := |D| \cdot \sum_{k \in K} k$, $\bar{D} := \max_{k,d}(k-1)d + 1$. A detailed analysis is provided in Appendix A.2.

Method	MULTs	ADDs
mixed-results (Eqn. 2)	$(c_{in}c_{out}\bar{K} + c_{out}\lvert K\rvert\lvert D\rvert)n$	$(c_{in}c_{out}\bar{K} + c_{out}\lvert K\rvert\lvert D\rvert)n$
mixed-weights (Eqn. 3)	$c_{in}c_{out}(\bar{K} + \bar{D}n)$	$c_{in}c_{out}\bar{D}(\lvert K\rvert\lvert D\rvert + n)$
DASH (Eqn. 4)	$c_{in}c_{out}(\bar{K} + n) + \mathcal{O}(c_{in}c_{out}n \log n)$	$c_{in}c_{out}(\lvert K\rvert\lvert D\rvert\bar{D} + n) + \mathcal{O}(c_{in}c_{out}n \log n)$

Here $\{\mathbf{w}_{k,d} | (k,d) \in K \times D\}$ are the kernel weights. The resulting supernet can be trained end-to-end, and our hope is that after search, the most important operation is assigned the highest weight. However, the complexity of computing the above summation directly, a baseline algorithmic approach we call ***mixed-results***, is $\mathcal{O}(c_{in}c_{out}(|K||D| + \bar{K})n)$, where $\bar{K} := |D| \sum_{k \in K} k$. *Mixed-results* can be expensive when we increase the maximum element in K or D with larger input size n. To improve upon it, we propose three techniques which build up to the efficiency-oriented DASH.

3.2.1 Technique 1: Mixed-Weights

Since convolution is linear, instead of computing $|K||D|$ convolutions, we can combine the kernels and compute convolution once. We call this approach ***mixed-weights***:

$$\mathbf{AggConv}_{K,D}(\mathbf{x}) = \mathbf{Conv}\left(\sum_{k \in K}\sum_{d \in D} \alpha_{k,d} \cdot \mathbf{w}'_{k,d}\right)(\mathbf{x}). \tag{3}$$

Here $\mathbf{w}'$ is the properly padded version of $\mathbf{w}$ (appending 0's at the end of each dimension) so that filters of different sizes can be added. The aggregated kernel has size $\bar{D} := \max_{k,d}(k-1)d + 1$ and the n-dependent term of the complexity of *mixed-weights* is $c_{in}c_{out}\bar{D}n$. Hence, it removes the direct dependence of the leading-order term on $|K||D|$, the search space size, that *mixed-results* had.

3.2.2 Technique 2: Fourier Convolution

If we wish to increase the kernel size and dilation with the input size, the complexity of *mixed-weights* will still grow *implicitly* with the search space size through the dependence on $\bar{D}$. To address this issue, we combine the kernel re-weighting idea with another technique motivated by the convolution theorem. Given a kernel $\mathbf{w}$, recall that $\mathbf{Conv}(\mathbf{w})(\mathbf{x}) = \mathbf{F}^{-1}\,\mathrm{diag}(\mathbf{F}\mathbf{w}')\mathbf{F}\mathbf{x}$, where $\mathbf{F}$ is the discrete Fourier transform (DFT) and $\mathrm{diag}(\mathbf{z})$ is a diagonal matrix with entries $\mathbf{z}$. In other words, convolution involves multiplying the DFT of the kernel by that of the input. Since the DFT can be applied in time $\mathcal{O}(n \log n)$ using the Fast Fourier Transform (FFT) and apart from that we only need element-wise multiplication, this yields an efficient approach to reducing dependence on the combined kernel size $\bar{D}$. While Roberts et al. [7] replaced the DFTs with a *continuous* set of matrices for XD-operations, our approach can be viewed as replacing the middle DFT with a *discrete* set of matrices for the purpose of efficiency. Accordingly, DASH computes $\mathbf{AggConv}_{K,D}$ as follows:

$$\mathbf{AggConv}_{K,D}(\mathbf{x}) = \mathbf{F}^{-1}\,\mathrm{diag}\left(\mathbf{F}\left(\sum_{k \in K}\sum_{d \in D} \alpha_{k,d} \cdot \mathbf{w}'_{k,d}\right)\right)\mathbf{F}\mathbf{x}. \tag{4}$$

Note that while the kernel changes for each $\mathbf{Conv}_{k,d}$, the input does not. Hence, we also save time by transforming the input to the frequency domain only once.

In Table 1, we report the theoretical complexities of the baselines and DASH, the latter leveraging both Technique 1 and 2. It is easy to obtain the operation complexities of *mixed-weights* and *mixed-results*. For DASH, the number of multiplications and additions can be attributed to the inner weight sum and multi-channel product (the first term) as well as three FFTs (the second). A detailed analysis is provided in Appendix A.2. We see that *mixed-weights* is favorable to *mixed-results* when $\bar{D} < \bar{K} = \mathcal{O}(|D|k_{\max}^2)$, which occurs with large kernels and a few dilations. On the other hand, only DASH completely separates the dominant terms containing $c_{in}c_{out}n$ from the size of the search space and its elements, replacing them by $\mathcal{O}(\log n)$, which is small for any realistic n. As we increase $k_{\max}$ and $d_{\max}$ for larger inputs, this will also lead to a slower asymptotic increase in complexity, making DASH an attractive choice for the multi-scale search space where $\bar{D}$ is large by design to extract possible long-range dependencies in the data.

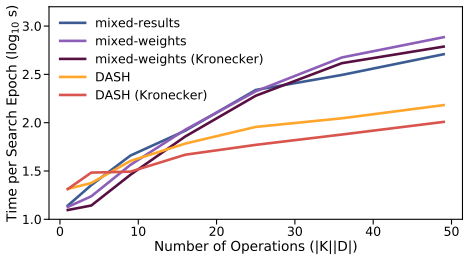


Figure 2: $\log_{10}$ time for one search epoch vs. number of ops in $S_{\mathbf{AggConv}_{K,D}}$. We vary the search space by letting $K = \{2p+1 | 1 \leq p \leq c\}$, $D = \{2^q - 1 | 1 \leq q \leq c\}$ and increasing c from 1 to 7.

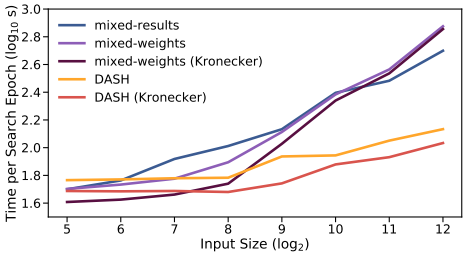


Figure 3: $\log_{10}$ time for one search epoch vs. input length of single-channel 1D data. We fix $K = \{3, 5, 7, 9, 11\}$, $D = \{1, 3, 7, 15, 31\}$ and test $n \in \{2^5, \ldots, 2^{12}\}$.

3.2.3 Technique 3: Kronecker Dilation

To efficiently implement the kernel summation in *mixed-weights* and DASH on a GPU, we introduce our final technique: after initializing $\mathbf{w}_{k,d}$ for each $\mathbf{Conv}_{k,d}$ separately, we use a Kronecker product $\otimes$ to transform the undilated kernels into dilated forms. For example, to compute a 2D convolution with dilation d, we introduce the sparse pattern matrix $\mathbf{P} \in \mathbb{R}^{d \times d}$ whose entries are all 0's except for the upper-left entry $\mathbf{P}_{1,1} = 1$. Then, $\mathbf{w}_{k,d} = \mathbf{w}_{k,1} \otimes \mathbf{P}$. Beyond the theoretical gains shown in Wu et al. [35], this dilation strategy is empirically faster than the standard way of padding 0's into $\mathbf{w}_{k,1}$ (Fig. 2 and 3). After dilating the kernels, we sum them together, zero-pad to match the input size, and apply the FFTs.

3.2.4 Ablation Study for the Proposed Techniques

To check that our asymptotic analysis leads to actual speedups and perform an ablation study on the proposed techniques, we evaluate the three methods on single-channel 1D input (experiment details are in Appendix A.3). Since both *mixed-weights* and DASH require kernel summation, which can be implemented with Kronecker dilation (Technique 3), we compare five methods in total.

Fig. 2 illustrates the combined forward- and backward-pass time in log scale for one search epoch vs. the size of $S_{\mathbf{AggConv}_{K,D}}$ when $n = 1000$. For small $\bar{D}$, the FFT overhead makes DASH runtime slightly longer but the difference is negligible. However, as $\bar{D}$ increases, the DASH curves grow much slower whereas the runtimes for the other methods scale with the number of operations. In Fig. 3, we fix K and D to study how runtime is affected by input size n. Both *mixed-results* and *mixed-weights* become extremely inefficient for large n's which commonly occur in time-series or signal processing. Surprisingly, DASH's runtime does not increase much with n. We hypothesize that this is due to wallclock-time being dominated by data-passing at that speed.

In general, Technique 1 on its own scales poorly for the considered search space. This is why methods like MergeNAS [27] cannot be used in our setting. Though XD makes use of Technique 2, it considers a parametrized space with infinitely many operations that need to be continuously evolved and is too expensive to be applied to tasks beyond CIFAR-100 (c.f. Fig. 1b). Technique 3 contributes to $2\times$ speedups for both *mixed-weights* and DASH. Overall, DASH (Kronecker) leads to about $10\times$ search-time speedups compared to the *mixed-results* scheme of DARTS for both the large operation space and large input size regimes. Hence, we use this version of DASH in later experiments.

3.3 The Full Pipeline: Architecture Search, Hyperparameter Optimization, and Retraining

Having shown the main techniques for searching a large space of kernel patterns, we now detail the full search and model development pipeline. Given a dataset, we set $K = \{3 + d(p-1) | 1 \leq p \leq p_{\max}\}$ for kernel sizes and $D = \{2^q - 1 | 1 \leq q \leq q_{\max}\}$ for dilations. For 2D input, we set d to 2, $p_{\max}$ to 4, and $q_{\max}$ to 4. For longer 1D sequence data, we set $d = 4$, $p_{\max} = 5$, and $q_{\max} = 4$. For instance, CIFAR has size $3 \times 32 \times 32$ where 3 is the number of channels and $n = 32$. The corresponding K is $\{3, 5, 7, 9\}$ and D is $\{1, 3, 7, 15\}$. To normalize architecture parameters into a probability distribution, we adopt the soft Gumbel Softmax activation, similar to Xie et al. [18].

The backbone networks are as follows. For 2D tasks, we use WRN 16-1 as the search backbone to accelerate supernet training and WRN 16-4 for retraining. For 1D tasks, we use 1D WRN [34] in the entire pipeline. During search, we subsample the training data at each epoch. Given the loss for the

target task, DASH jointly optimizes the model weights and the architecture parameters using direct gradient descent. This single-level optimization is more efficient than two-stage NAS, which finds initial assignments for architecture parameters and trains the candidates from scratch.

After searching for a predefined number of epochs, we discretize the search space and pick $\mathbf{Conv}_{k,d} \in S_{\mathbf{AggConv}_{K,D}}$ with the largest weight for each layer. The final model has a similar overall structure to the backbone, but the intermediate operations are tailored to the target task. To improve training stability, we additionally add a simple hyperparameter tuning stage between search and retraining using grid search (configuration space shown in Appendix A.5.2). For each setting, we train the discretized model on *a subset* of the training data for *fewer* epochs so the tuning cost is a small fraction of the entire pipeline's cost (Table 6), Then, we evaluate the performance on a holdout validation set and select the configuration with the best validation score. As a final step, we retrain the discretized model with the optimal hyperparameters on all training data until convergence. Like other weight-sharing methods with discretization, our final model will be more efficient than the supernet.

4 Evaluation

We evaluate the performance of DASH on diverse tasks using ten datasets from NAS-Bench-360 [4], a benchmark spanning multiple application domains, input dimensions, and learning objectives.[2] These include classical vision tasks such as CIFAR-100 where CNNs do well, scientific computing tasks such as Darcy Flow where standard CNN backbones can perform poorly [7, 36], sequence tasks such as DeepSEA where large dilations are preferred [14, 26], and many others. Thus, our evaluation will not only test whether DASH can find good architectures in the proposed new search space, but also investigate whether multi-scale convolution is a strong competitor for solving different problems. In fact, our results show that DASH is a top choice for many tasks, obtaining in-aggregate the best speed-accuracy trade-offs among the methods we evaluate (c.f. Fig. 5).

4.1 Baselines and Experimental Setup

For each NAS-Bench-360 task, we compare DASH with the following methods: DenseNAS [5] and GAEA PC-DARTS [3], which represent general NAS; Auto-DeepLab [37] and AMBER [26], which represent specialist NAS methods for dense prediction and 1D tasks, respectively; 1D temporal convolutional network (TCN) [14], regular WRN, and WRN with hyperparameter tuner ASHA [38], which represent natural NAS baselines; and Perceiver IO [31], which represents non-NAS general-purpose models. While these results are available in Tu et al. [4], we additionally add a BABY DASH baseline: we run DASH in the DARTS convolution space with $K = \{3, 5\}$ and $D = \{1, 2\}$ to study whether large kernel sizes and dilations are necessary to strong performance across-the-board. Finally, we compare our method to the expert architectures selected by NAS-Bench-360. These models are representatives of the best that hand-crafting has to offer.

Each dataset is preprocessed and split using the NAS-Bench-360 script, with the training set being used for search, hyperparameter tuning, and retraining. To construct the multi-scale search space, we set K and D according to the rules in Section 3.3. We use the default SGD optimizer for the WRN backbone and fix the learning rate schedule as well as the gradient clipping threshold for every task. The entire DASH pipeline can be run on a single NVIDIA V100 GPU, which is also the system that we use to report the runtime cost. Full experimental details can be found in the Appendix.

We evaluate the performance of all competing methods following the NAS-Bench-360 protocol. We first report results of the target metric for each task by running the model of the *last* epoch on the test data. Then, we report aggregate results via *performance profiles* [39], a technique that considers both outliers and small performance differences to compare methods across multiple tasks robustly. In such plots, each curve represents one method. The τ on the x-axis denotes the fraction of tasks on which a method is no worse than a τ-factor from the best.

4.2 Results and Discussion

We present the accuracy results for each task in Table 2 and the performance profiles in Fig. 4. Fig. 4 clearly demonstrates that DASH is superior to other competing methods in terms of aggregate performance. In particular, it achieves top performance for 6/10 tasks, ranks first among all *automated* models for 7/10 tasks, and performs favorably when considering both accuracy and efficiency as

[2]For completeness, we give a task summary in the Appendix.

Table 2: Accuracy results on NAS-Bench-360 tasks across diverse application domains and problem dimensions (the last three problems are 1D and the rest are 2D). DASH beats *all the other NAS methods* on 7/10 tasks and exceeds hand-designed expert models on 6/10 tasks. Scores of DASH are averaged over three trials. Scores of the baselines are from Tu et al. [4]. See Table 5 in the Appendix A.6 for standard deviations.

	CIFAR-100 0-1 error ^{l}	Spherical 0-1 error ^{l}	Darcy Flow relative ℓ_2 ^{l}	PSICOV MAE_8 ^{l}	Cosmic FNR ^{l}	NinaPro 0-1 error ^{l}	FSD50K mAP ^{h}	ECG F1 ^{h}	Satellite 0-1 error ^{l}	DeepSEA AUROC ^{h}
Expert	**19.39**	67.41	0.008	3.35	25.29	8.73	0.38	**0.72**	19.8	0.70
WRN	23.35	85.77	0.073	3.84	51.76	6.78	0.08	0.57	15.49	0.60
TCN	-	-	-	-	-	-	-	0.43	16.21	0.56
WRN-ASHA	23.39	75.46	0.066	3.84	37.53	7.34	0.09	0.57	15.84	0.59
DARTS-GAEA	24.02	**48.23**	0.026	**2.94**	31.15	17.67	0.06	0.66	12.51	0.64
DenseNAS	25.98	72.99	0.10	3.84	79.52	10.17	0.36	0.60	13.81	0.60
Auto-DL	-	-	0.049	6.73	99.79	-	-	-	-	-
AMBER	-	-	-	-	-	-	-	0.67	12.97	0.68
Perceiver IO	70.04	82.57	0.24	-	-	22.22	0.28	0.34	15.93	0.62
BABY DASH	25.56	68.55	0.016	3.94	**13.18**	8.28	0.38	0.63	13.29	0.63
DASH	24.37	75.44	**0.0079**	3.30	15.04	**6.60**	**0.40**	0.68	**12.28**	**0.72**

$^{h/l}$ a higher / lower value of the metric indicates better performance.

shown in Fig. 5. In addition, Table 3 shows that DASH outperforms DARTS in speed for all 10 tasks (in several cases by an order of magnitude), and attains comparable efficiency with training vanilla WRNs for 6/10 tasks (full-pipeline time is less than or about twice as long as the WRN training time). In the following, we provide a detailed analysis of the experimental results.

DASH dominates automated methods. Compared to other automated methods, DASH has a clear advantage in accuracy. Even for tasks where it does not beat the expert, e.g. ECG, DASH's performance is significantly better than other AutoML methods. It also outperforms specialist methods Auto-DL and AMBER on dense prediction and 1D tasks, respectively. Although DARTS does best on CIFAR-100 (the task for which it was designed), Spherical, and PSICOV, it is the worst on NinaPro and FSD50K. Note that the underperformance of DASH on CIFAR-100 relative to WRN (and on Spherical and Cosmic relative to BABY DASH) suggests suboptimality of the gradient descent optimization procedure but not of the operation space, since WRN and BABY DASH are contained in our search space. This indicates a future direction to improve optimization in the DASH search space.

DASH dominates expert architectures. While the degree of sophistication of the expert networks varies task by task, the performance of DASH on tasks such as Darcy Flow suggests that it is capable of competing with highly specialized networks, e.g., Fourier Neural Operator [36] for PDE solving. These results imply that DASH, and more generally the strategy of equipping backbone networks with task-specific kernels, is a promising approach for tackling model development in new domains. Meanwhile, DASH consistently outperforms Perceiver IO which represents non-automated general-purpose models. We speculate that the poor performance of Perceiver IO is because it is developed on more sophisticated natural language and multi-modal reasoning tasks and training Transformers is generally difficult.

Table 3: Full-pipeline runtime in GPU hours for NAS-Bench-360 (PSICOV results are omitted due to a discrepancy in the implementation of data loading). DASH is consistently faster than DARTS, and it is less than a factor of two slower than simply training a WRN on six of the ten tasks. DenseNAS is fast but its accuracy is far less impressive. XD is too expensive to be applied to tasks other than CIFAR-100 [7].

Task	DARTS	DenseNAS	WRN	DASH
CIFAR-100	9.5	2.5	2	2.5
Spherical	16.5	2.5	2	5
Darcy Flow	6.5	0.5	0.5	5.3
Cosmic	21.5	2.5	4	6.8
NinaPro	0.5	0.2	0.2	0.3
FSD50K	37	4.5	4	29
ECG	140	6.5	5	1.3
Satellite	28	3	4.5	6.5
DeepSEA	39.5	2	1.5	10

Large kernels are needed. We also ablate the large-k-large-d design of the search space by comparing DASH with BABY DASH. We hypothesize that for the same task, a small performance gap between the two methods would indicate that small kernels suffice for extracting local features, whereas a major degradation in the quality of the BABY DASH model can imply that the task needs global modeling. Consequently, datasets such as Darcy Flow and ECG provide compelling evidence that kernels with large receptive

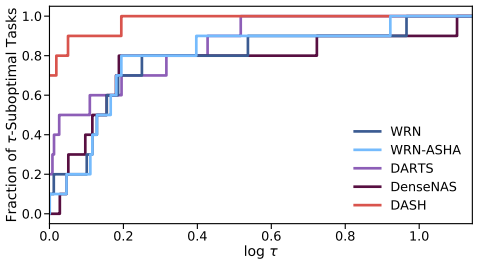


Figure 4: Performance profiles of general NAS methods and DASH on NAS-Bench-360. DASH being far in the top left corner indicates it is rarely suboptimal and is often the best.

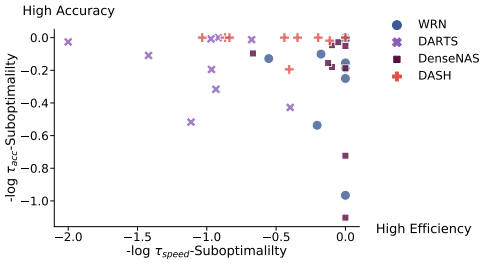


Figure 5: Comparing $-\log \tau$-suboptimality of speed vs. accuracy on all tasks. DASH's concentration in the top right corner indicates its strong efficacy-efficiency trade-offs relative to the other methods.

fields play an important role in solving real-life problems and further back up the design of our multi-scale convolutional search space. An example of the series of WRN kernels found by DASH on Darcy Flow is: $\mathbf{Conv}_{5,3} \to \mathbf{Conv}_{3,1} \to \mathbf{Conv}_{3,1} \to \mathbf{Conv}_{3,15} \to \mathbf{Conv}_{7,15} \to \mathbf{Conv}_{9,7} \to \mathbf{Conv}_{9,7} \to \mathbf{Conv}_{3,7} \to \mathbf{Conv}_{5,7} \to \mathbf{Conv}_{5,15} \to \mathbf{Conv}_{9,7} \to \mathbf{Conv}_{3,7} \to \mathbf{Conv}_{7,7}$. We can see that large kernels are indeed selected during search. More visualizations can be found in Appendix A.8.

DASH is computationally efficient. In addition to prediction quality, we also care about the efficiency of model selection. Table 3 provides the combined search and retraining time in GPU hours for DARTS, DenseNAS, and DASH, as well as the training time for vanilla WRN 16-4 without hyperparameter tuning (baseline results are taken from Tu et al. [4]). We also present the breakdown of DASH's full-pipeline runtime in Appendix A.7. A key observation is that the cost of DASH is consistently below DARTS' on all tasks and is similar to training a simple CNN for more than half of them. Although DenseNAS stands out by speed, its practical performance is less impressive.

In Fig. 5, we visualize the trade-off between efficiency and effectiveness for each method and task combination. Evidently, DASH takes an important step towards bridging the gap between the efficiency of DARTS and the expressivity of XD in NAS. The fact that DASH can be trained at a low cost testifies that we need not sacrifice efficiency for adding more operations. In fact, we have actually shown that DASH is *both* faster *and* more effective than DARTS in many tasks.

4.3 Limitations and Future Work

There are several open problems which we leave for future work. First, it is beneficial to study why certain kernel patterns are chosen, as the selected operations can hint us at the intrinsic properties of the datasets. Second, although this paper focuses on NAS, which is an alternative to fine-tuning pretrained models, the aggregated convolution can be a plug-and-play module for algorithms that search for large-scale models. Third, one can improve upon DASH itself, e.g., by including non-square convolutions for 2D problems or using a better optimization algorithm. More ambitiously, one could construct a more comprehensive search space containing high-level operators such as self-attention [40] or use other backbones such as ConvNeXt [15].

Societal Impact This paper calls for ML community's attention to less-studied application domains and moves towards truly democratizing machine learning in real life. In terms of broader societal impact, our work can exert a positive influence as it contributes to NAS efficiency and reduces the computational burden on AutoML end-users. However, lowering the barrier for applying ML to a wide range of tasks necessarily comes with the risk of misuse. Hence, it is imperative to develop NAS methods with privacy, safety, and fairness guarantees.

5 Conclusion

In this paper, we argue that a crucial goal of NAS is to discover accessible models for diverse tasks. To this end, we propose DASH, which efficiently searches for convolution patterns and integrates them into existing backbones. DASH overcomes the computational limitations of differentiable NAS and obtains high-quality models with accuracy comparable to or better than that of the handcrafted networks on many tasks. Our experiments show that convolution can be a universal operator for many under-explored areas. DASH is also a promising step towards developing general-purpose models with more complicated structures.

References

[1] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[2] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[3] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[4] Renbo Tu, Mikhail Khodak, Nicholas Roberts, and Ameet Talwalkar. NAS-Bench-360: Benchmarking diverse tasks for neural architecture search. arXiv, 2021.

[5] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[6] Esteban Real, Chen Liang, David R. So, and Quoc V. Le. AutoML-Zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[7] Nicholas Roberts, Mikhail Khodak, Tri Dao, Liam Li, Chris Ré, and Ameet Talwalkar. Rethinking neural operations for diverse tasks. In *Advances in Neural Information Processing Systems*, 2021.

[8] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Arxiv*, 2016.

[9] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. In *International Conference on Learning Representations*, 2017.

[10] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. 2016.

[11] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin P. Murphy, and Alan Loddon Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40: 834–848, 2018.

[12] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters — improve semantic segmentation by global convolutional network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1743–1751, 2017.

[13] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[14] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv, 2018.

[15] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. 2022.

[16] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[17] 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*, 2018.

[18] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[19] 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. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[20] M. Suganuma, M. Ozay, and T. Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In *International Conference on Machine Learning*, 2018.

[21] Abhinav Mehrotra, Alberto Gil C. P. Ramos, Sourav Bhattacharya, Lukasz Dudziak, Ravichander Vipperla, Thomas C. P. Chau, Mohamed Saleh Abdelfattah, Samin S. Ishtiaq, and Nicholas D. Lane. Nas-bench-asr: Reproducible neural architecture search for speech recognition. In *International Conference on Learning Representations*, 2021.

[22] David R. So, Chen Liang, and Quoc V. Le. The evolved transformer. In *International Conference on Machine Learning*, 2019.

[23] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *International Conference on Machine Learning*, 2016.

[24] Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. 2016.

[25] Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, 2018.

[26] Zijun Zhang, Christopher Y. Park, Chandra L. Theesfeld, and Olga G. Troyanskaya. An automated framework for efficiently designing deep convolutional neural networks in genomics. *Nature Machine Intelligence*, 3(5):392–400, 2021.

[27] Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. Mergenas: Merge operations into one for differentiable architecture search. In *IJCAI*, 2020.

[28] Xiaohan Ding, X. Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13728–13737, 2021.

[29] Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[30] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, J. Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *ECML/PKDD*, 2019.

[31] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver io: A general architecture for structured inputs outputs, 2022.

[32] Kevin Lu, Aditya Grover, P. Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines. *ArXiv*, abs/2103.05247, 2021.

[33] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016.

[34] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and Franccois Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34:1936–1962, 2020.

[35] Tianyi Wu, Sheng Tang, Rui Zhang, Juan Cao, and Jintao Li. Tree-structured kronecker convolutional network for semantic segmentation. *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 940–945, 2019.

[36] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[37] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[38] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. 2020.

[39] Elizabeth D. Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[41] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[42] Taco Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. In *International Conference on Machine Learning*, 2018.

[43] David Josephs, Carson Drake, Andrew M Heroy, and John Santerre. semg gesture recognition with a simple model of attention. *Machine Learning for Health*, pages 126–138, 2020.

[44] Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, and Xavier Serra. Fsd50k: an open dataset of human-labeled sound events. *ArXiv*, abs/2010.00475, 2021.

[45] Badri Adhikari. DEEPCON: protein contact prediction using dilated convolutional neural networks with dropout. *Bioinformatics*, 36(2):470–477, 07 2019.

[46] Keming Zhang and Joshua S. Bloom. deepcr: Cosmic ray rejection with deep learning. *The Astrophysical Journal*, 889(1):24, 2020.

[47] Shenda Hong, Yanbo Xu, Alind Khare, Satria Priambada, Kevin O. Maher, Alaa Aljiffry, Jimeng Sun, and Alexey Tumanov. Holmes: Health online model ensemble serving for deep learning models in intensive care units. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[48] Angus Dempster, Franccois Petitjean, and Geoffrey I. Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34:1454–1495, 2020.

[49] Jian Zhou and Olga G. Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature Methods*, 12:931–934, 2015.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Our work aims to democratize machine learning in diverse real-life applications. However, reducing the barrier for deploying accurate ML models on a diverse range of tasks may introduce unintended consequences associated with safety/fairness/privacy issues. We have addressed this point in Section 4.3.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Complexity analysis for Table 1 is provided in Appendix A.2.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Due to limited space, we present the error bars in Appendix A.6.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See Table 4.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The data we used are open-sourced and our usage comply with their terms.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The data we used do not include any personally identifiable information.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Term Clarification

Since we compare with a variety of methods in the paper, here we clarify some of the terms we use.

what we say	what we are referring to
Best-Automated (Fig. 1a)	WRN, WRN-ASHA, DARTS, DenseNAS, Auto-DL, AMBER
Hand-Designed (Fig. 1a)	Expert architectures in Table 4
AutoML	WRN-ASHA, DARTS, DenseNAS, Auto-DL, AMBER
NAS	DARTS, DenseNAS, Auto-DL, AMBER
WRN	WRN without hyperparameter tuning

A.2 Asymptotic Analysis

In this section we outline the runtime analysis used to populate the asymptotic complexities in Table 1. All three methods in the table—*mixed-results*, *mixed-weights*, and DASH—are computing the following weighted sum of convolutions:

$$\mathbf{AggConv}_{K,D}(\mathbf{x}) := \sum_{k \in K} \sum_{d \in D} \alpha_{k,d} \cdot \mathbf{Conv}(\mathbf{w}_{k,d})(\mathbf{x}). \tag{5}$$

We consider 1D inputs $\mathbf{x}$ with length n and c_{in} input channels; the convolutions have c_{out} output channels. We view $\mathbf{Conv}(\mathbf{w}_{k,d})(\mathbf{x})$ as having the naive complexity $c_{in}c_{out}kn$ since the deep learning frameworks use the direct (non-Fourier) algorithm. *mixed-results* computes the sum directly, which involves (1) applying one convolution of each size k and dilation to $\mathbf{x}$ at a cost of $c_{in}c_{out}kn$ MULTs and ADDs each for a total cost of $c_{in}c_{out}\bar{K}n$, (2) scalar-multiplying the outputs at a cost of $c_{out}|K||D|n$ MULTs, and (3) summing the results together at a cost of $c_{out}|K||D|n$ ADDs. *mixed-weights* instead (1) multiplies all kernels by their corresponding weight at a cost of $c_{in}c_{out}\bar{K}$ MULTs, (2) zero-pads the results to the largest effective kernel size $\bar{D}$ and adds them together at a cost of $c_{in}c_{out}|K||D|\bar{D}$ ADDs, and (3) applies the resulting $\bar{D}$-size convolution to the input at a cost of $c_{in}c_{out}\bar{D}n$ MULTs and ADDs. Finally, DASH also (1) does the first two steps of *mixed-weights* at a cost of $c_{in}c_{out}\bar{K}$ MULTs and $c_{in}c_{out}|K||D|\bar{D}$ ADDs but then (2) pads the resulting $\bar{D}$-size convolution to size n and applies an FFT at a cost of $\mathcal{O}(c_{in}c_{out}n\log n)$ MULTs and ADDs, (3) applies an FFT to $\mathbf{x}$ at a cost of $\mathcal{O}(c_{in}n\log n)$, (4) element-wise multiplies the transformed filters by the inputs at a cost of $c_{in}c_{out}n$ MULTs, (5) adds up c_{in} results for each of c_{out} output channels at a cost of $c_{in}c_{out}$ MULTs, and (6) applies an iFFT to the result at a cost of $\mathcal{O}(c_{out}n\log n)$.

A.3 Experiment Details for Fig. 2 and Fig. 3

For the speed tests, we work with the Sequential MNIST dataset, i.e., the 2D 28×28 images are stretched into 1D with length 784. We zero pad or truncate the input to generate data with different input size n. The backbone is 1D WRN with the same structure as introduced in Section 3. The batch size is 128. We run the workflow on a single NVIDIA V100 GPU. The timing results reported are the $\log_{10}$(combined forward and backward pass time for one search epoch).

In Fig. 2, we study how the size of our multi-scale convolution search space affects the runtimes of *mixed-results*, *mixed-weights*, and DASH for $n = 1000$ (zero-padded MNIST). We define $K = \{3 + 2(p-1) | 1 \leq p \leq c\}$, $D = \{2^q - 1 | 1 \leq q \leq c\}$ and varies c from 1 to 7. Consequently, the number of operations included in the search space grows from 1 to 49.

In Fig. 3, we study how the input size affects the runtimes of the three methods. We fix $K = \{3, 5, 7, 9, 11\}$, $D = \{1, 3, 7, 15, 31\}$ and vary n from 2^5 to 2^{12}.

A.4 Information About Tasks in NAS-Bench-360

Table 4: Information about evaluation tasks in NAS-Bench-360 [4].

Task name	# Data	Data dim.	Type	License	Learning objective	Expert arch.
CIFAR-100	60K	2D	Point	CC BY 4.0	Classify natural images into 100 classes	DenseNet-BC [41]
Spherical	60K	2D	Point	CC BY-SA	Classify spherically projected images into 100 classes	S2CN [42]
NinaPro	3956	2D	Point	CC BY-ND	Classify sEMG signals into 18 classes corresponding to hand gestures	Attention Model [43]
FSD50K	51K	2D	Point (multi-label)	CC BY 4.0	Classify sound events in log-mel spectrograms with 200 labels	VGG [44]
Darcy Flow	1100	2D	Dense	MIT	Predict the final state of a fluid from its initial conditions	FNO [36]
PSICOV	3606	2D	Dense	GPL	Predict pairwise distances between residuals from 2D protein sequence features	DEEPCON [45]
Cosmic	5250	2D	Dense	Open License	Predict propablistic maps to identify cosmic rays in telescope images	deepCR-mask [46]
ECG	330K	1D	Point	ODC-BY 1.0	Detect atrial cardiac disease from a ECG recording (4 classes)	ResNet-1D [47]
Satellite	1M	1D	Point	GPL 3.0	Classify satellite image pixels' time series into 24 land cover types	ROCKET [48]
DeepSEA	250K	1D	Point (multi-label)	CC BY 4.0	Predict chromatin states and binding states of RNA sequences (36 classes)	DeepSEA [49]

A.5 Evaluation of DASH on NAS-Bench-360

A.5.1 Backbone Network Structure

2D Tasks We use the Wide ResNet 16-4 [33] as the backbone for all 2D tasks. The original model is made up of 16 3×3 conv followed by 6 WRN blocks with the following structure ($i \in \{1, 2, 3, 4, 5, 6\}$ indicates the block index):

BatchNorm, ReLU	
Conv 1	$16 \times 4 \times ((i+1)//2)$ $(k = 3, d = 1)$ filters, ReLU
Dropout	dropout rate p
BatchNorm, ReLU	
Conv 2	$16 \times 4 \times ((i+1)//2)$ $(k = 3, d = 1)$ filters , stride = $(i+1)//2$, ReLU
Add residual (apply point-wise conv first if $c_{in} \neq c_{out}$)	

The output block consists of a BatchNorm layer, a ReLU activation, a linear layer, and a final activation layer which we modify according to the task learning objective, e.g., log softmax for classification and sigmoid for dense prediction. We set $p = 0$ in search and tune p as a hyperparameter for retraining. We use the WRN code provided here: `https://github.com/meliketoy/wide-resnet.pytorch`.

1D Tasks We use the 1D WRN [34] as the backbone for all 1D tasks. The model is made up of 3 residual blocks with the following structure:

Conv 1	c_{out} $(k = 8, d = 1)$ filters
Dropout	dropout rate p
BatchNorm, ReLU	
Conv 2	c_{out} $(k = 5, d = 1)$ filters
Dropout	dropout rate p
BatchNorm, ReLU	
Conv 3	c_{out} $(k = 3, d = 1)$ filters
Dropout	dropout rate p
BatchNorm, ReLU	

In the original architecture, $c_{out} = 64$. We set c_{out} to $\min(4^{\text{num_classes}//10+1}, 64)$ to account for simpler tasks with fewer class labels. The output block consists of a linear layer and a activation layer which we modify according to the task learning objective, e.g., log softmax for classification and sigmoid for dense prediction. We set $p = 0$ in search and tune p as a hyperparameter for retraining. We use the 1D WRN code provided here: `https://github.com/okrasolar/pytorch-timeseries`.

A.5.2 DASH Pipeline Hyperparameters

Search

- Epoch: 100
- Optimizer: `SGD(momentum=0.9, nesterov=True, weight_decay=5e-4)` for both model weights and architecture parameters
- Model weight learning rate: 0.1 for point prediction tasks, 0.01 for dense tasks
- Architecture parameter learning rate: 0.05 for point prediction tasks, 0.005 for dense tasks
- Learning rate scheduling: decay by 0.2 at epoch 60
- Gradient clipping threshold: 1
- Softmax temperature: 1
- Subsampling ratio: 0.2

Hyperparameter tuning

- Epoch: 80
- Configuration space:
  - Learning rate: {1e-1, 1e-2, 1e-3}
  - Weight decay: {5e-4, 5e-6}
  - Momentum: $\{0.9, 0.99\}$
  - Dropout rate: $\{0, 0.05\}$

Retraining

- Epoch: 200
- Learning rate scheduling: for 2D tasks, decay by 0.2 at epoch 60, 120, 160; for 1D tasks, decay by 0.2 at epoch 30, 60, 90, 120, 160

Task-Specific Hyperparameters

2D tasks	CIFAR-100	Spherical	Darcy Flow	PSICOV	Cosmic	NinaPro	FSD50K
Batch size	64	64	10	8	4	128	128
Input size	(32, 32)	(60, 60)	(85, 85)	(128, 128)	(128, 128)	(16, 52)	(96, 101)
Kernel sizes (K)	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$
Dilations (D)	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$
Loss (l)	Cross Entropy	Cross Entropy	L2	MSE	BCE	Focal	BCE w. Logits

1D tasks	Satellite	ECG	DeepSEA
Batch size	256	1024	256
Input size	46	1000	1000
Kernel sizes (K)	$\{3, 7, 11, 15, 19\}$	$\{3, 7, 11, 15, 19\}$	$\{3, 7, 11, 15, 19\}$
Dilations (D)	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$
Loss (l)	Cross Entropy	Cross Entropy	BCE w. Logits

A.6 Accuracy Results on NAS-Bench-360 with Error Bars

Table 5: Performance of DASH and the baselines on tasks in NAS-Bench-360. Methods are grouped into three classes: non-automated, automated, and the DASH family. Results of DASH are averaged over three trials using the models obtained after the last retraining epoch. Results of baselines are taken from the leaderboard on `https://github.com/rtu715/NAS-Bench-360` at the time of submission.

	CIFAR-100 0-1 error ^{l}	Spherical 0-1 error ^{l}	Darcy Flow relative ℓ_2 ^{l}	PSICOV MAE_8 ^{l}	Cosmic FNR ^{l}
WRN	23.35±0.05	85.77±0.71	0.073±0.001	3.84±0.053	51.76±2.09
Expert	**19.39±0.20**	67.41±0.76	0.008±0.001	3.35±0.14	25.29±1.44
Perceiver IO	70.04±0.44	82.57±0.19	0.24±0.01	-	-
WRN-ASHA	23.39±0.01	75.46±0.40	0.066±0.00	3.84±0.05	37.53±10.16
DARTS-GAEA	24.02±1.92	**48.23±2.87**	0.026±0.001	**2.94±0.13**	31.15±3.48
DenseNAS	25.98±0.38	72.99±0.95	0.10±0.01	3.84±0.15	79.52±2.20
Auto-DL	-	-	0.049±0.005	6.73±0.73	99.79±0.02
BABY DASH	25.56±1.37	68.94±1.3	0.016±0.002	3.94±0.54	**13.18±1.88**
DASH	24.37±0.81	75.44±2.38	**0.0079±0.002**	3.30±0.16	15.04±2.35

	NinaPro 0-1 error ^{l}	FSD50K mAP ^{h}	ECG F1 score ^{h}	Satellite 0-1 error ^{l}	DeepSEA AUROC ^{h}
WRN	6.78±0.26	0.08±0.001	0.57±0.01	15.49±0.03	0.60±0.001
TCN	-	-	0.43±0.005	16.21±0.05	0.56±0.001
Expert	8.73±0.9	0.38±0.004	**0.72±0.00**	19.8±0.00	0.70±0.24
Perceiver IO	22.22±1.80	0.28±0.002	0.34±0.01	15.93±0.08	0.62±0.004
WRN-ASHA	7.34±0.76	0.09±0.03	0.57±0.01	15.84±0.52	0.59±0.002
DARTS-GAEA	17.67±1.39	0.06±0.02	0.66±0.01	12.51±0.24	0.64±0.02
DenseNAS	10.17±1.31	0.36±0.002	0.60±0.01	13.81±0.69	0.60±0.001
AMBER	-	-	0.67±0.015	12.97±0.07	0.68±0.01
BABY DASH	8.28±0.62	0.38±0.01	0.63±0.001	13.29±0.108	0.63±0.017
DASH	**6.60±0.33**	**0.40±0.008**	0.68±0.007	**12.28±0.5**	**0.72±0.013**

$^{h\,/\,l}$ a higher / lower value of the metric indicates better performance.

A.7 Runtime of DASH on NAS-Bench-360

Table 6: Runtime breakdown for DASH on NAS-Bench-360 tasks evaluated on a NVIDIA V100 GPU.

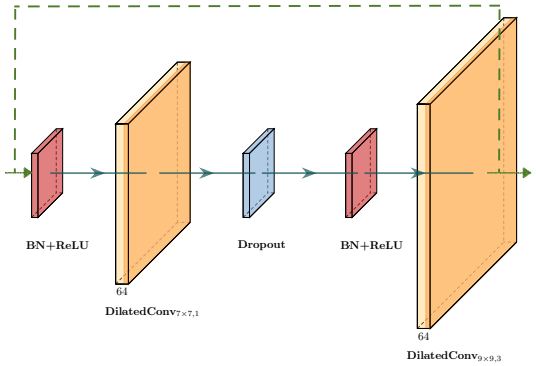
Task	Search	Hyperparameter Tuning	Retraining	Total
CIFAR-100	1.6	0.15	0.77	2.5
Spherical	1.6	0.25	3.16	5.0
Darcy Flow	0.16	1.6	3.5	5.3
PSICOV	0.88	0.64	14	15
Cosmic	1.6	0.055	5.1	6.8
NinaPro	0.028	0.16	0.11	0.30
FSD50K	0.88	0.88	27	29
ECG	0.18	0.28	0.83	1.3
Satellite	1.8	0.4	4.3	6.5
DeepSEA	0.36	1.6	8.3	10

A.8 Searched Architecture Visualization

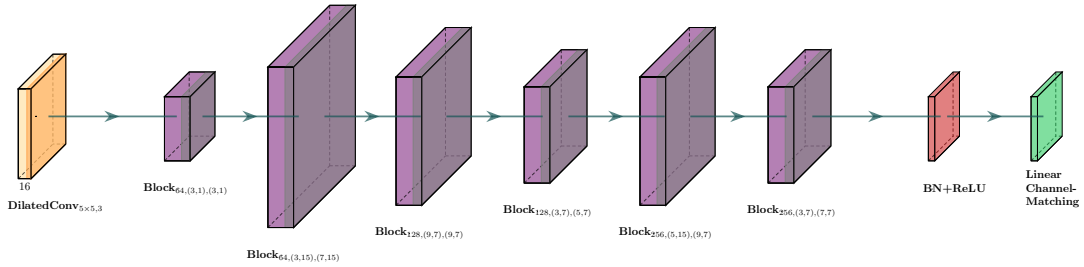
In this section, we give two example networks searched by DASH to show that large kernel matters for diverse tasks.

A.8.1 2D Example: Darcy Flow

For this problem, DASH generates a WRN 16-4 [33] for retraining. The network architecture consists of several residual blocks. For instance, we can use $\text{Block}_{64,(7,1),(9,3)}$ to denote the residual block with the following structure:



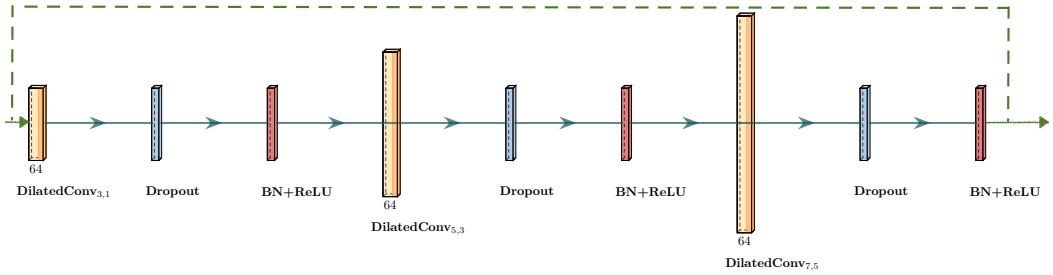
where 64 is the output channel and BN denotes the BatchNorm layer. Note that size of a convolutional layer in the figure is proportional to the kernel size but not the number of channels. Then, an example network produced by DASH for Darcy Flow looks like the following:



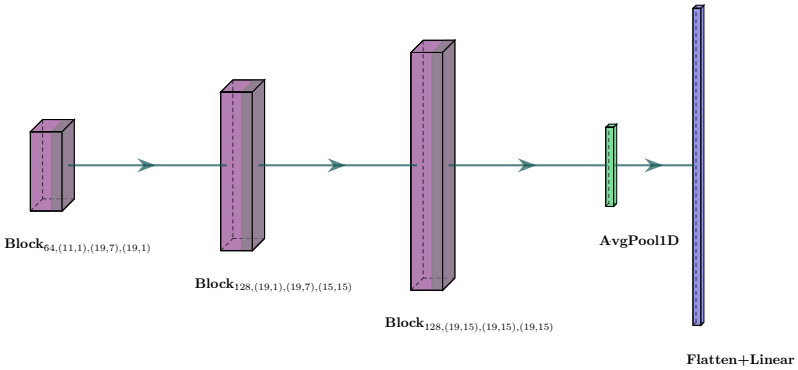
Since Darcy Flow is a dense prediction task, the last layer is a channel-matching (permutation+linear+permutation) layer instead of a pooling+linear layer for classification.

A.8.2 1D Example: DeepSEA

For this problem, DASH generates a 1D WRN [34] for retraining. The network architecture consists of several residual blocks. For instance, we can use $\text{Block}_{64,(3,1),(5,3),(7,5)}$ to denote the residual block with the following structure:



where 64 is the output channel and BN denotes the BatchNorm layer. Then, an example network produced by DASH for DeepSEA looks like the following:



We can see that large kernels are indeed selected during search.