Parameter Prediction for Unseen Deep Architectures

Boris Knyazev[1,2*] **Michal Drozdzal**[4,†] **Graham W. Taylor**[1,2,3,†] **Adriana Romero-Soriano**[4,5,†]

[1] University of Guelph [2] Vector Institute for Artificial Intelligence
[3] Canada CIFAR AI Chair [4] Facebook AI Research [5] McGill University
[†]equal advising

https://github.com/facebookresearch/ppuda

Abstract

Deep learning has been successful in automating the design of features in machine learning pipelines. However, the algorithms optimizing neural network parameters remain largely hand-designed and computationally inefficient. We study if we can use deep learning to directly predict these parameters by exploiting the past knowledge of training other networks. We introduce a large-scale dataset of diverse computational graphs of neural architectures – DEEPNETS-1M– and use it to explore parameter prediction on CIFAR-10 and ImageNet. By leveraging advances in graph neural networks, we propose a hypernetwork that can predict performant parameters in a *single forward pass* taking a fraction of a second, even on a CPU. The proposed model achieves surprisingly good performance on *unseen* and *diverse* networks. For example, it is able to predict all 24 million parameters of a ResNet-50 achieving a 60% accuracy on CIFAR-10. On ImageNet, top-5 accuracy of some of our networks approaches 50%. Our task along with the model and results can potentially lead to a new, more computationally efficient paradigm of training networks. Our model also learns a strong representation of neural architectures enabling their analysis.

1 Introduction

Consider the problem of training deep neural networks on large annotated datasets, such as ImageNet [1]. This problem can be formalized as finding optimal parameters for a given neural network a, parameterized by $\mathbf{w}$, w.r.t. a loss function $\mathcal{L}$ on the dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ of inputs $\mathbf{x}_i$ and targets y_i:

$$\arg\min_{\mathbf{w}} \sum\nolimits_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_i; a, \mathbf{w}), y_i), \tag{1}$$

where $f(\mathbf{x}_i; a, \mathbf{w})$ represents a forward pass. Equation 1 is usually minimized by iterative optimization algorithms – e.g. SGD [2] and Adam [3] – that converge to performant parameters $\mathbf{w}_p$ of the architecture a. Despite the progress in improving the training speed and convergence [4–7], obtaining $\mathbf{w}_p$ remains a bottleneck in large-scale machine learning pipelines. For example, training a ResNet-50 [8] on ImageNet can take many GPU hours [9]. With the ever growing size of networks [10] and necessity of training the networks repeatedly (e.g. for hyperparameter or architecture search), the classical process of obtaining $\mathbf{w}_p$ is becoming computationally unsustainable [11–13].

A new parameter prediction task. When optimizing the parameters for a *new* architecture a, typical optimizers disregard past experience gained by optimizing other nets. However, leveraging this past experience can be the key to reduce the reliance on iterative optimization and, hence the high computational demands. To progress in that direction, we propose a new task where iterative optimization is replaced with a *single forward pass* of a hypernetwork [14] $H_{\mathcal{D}}$. To tackle the task, $H_{\mathcal{D}}$ is expected to leverage the knowledge of how to optimize *other* networks $\mathcal{F}$. Formally, the

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

[*]Part of the work was done while interning at Facebook AI Research.

task is to predict the parameters of an *unseen* architecture $a \notin \mathcal{F}$ using $H_{\mathcal{D}}$, parameterized by θ_p: $\hat{\mathbf{w}}_p = H_{\mathcal{D}}(a; \theta_p)$. The task is constrained to a dataset $\mathcal{D}$, so $\hat{\mathbf{w}}_p$ are the predicted parameters for which the test set performance of $f(\mathbf{x}; a, \hat{\mathbf{w}}_p)$ is similar to the one of $f(\mathbf{x}; a, \mathbf{w}_p)$. For example, we consider CIFAR-10 [15] and ImageNet image classification datasets $\mathcal{D}$, where the test set performance is classification accuracy on test images.

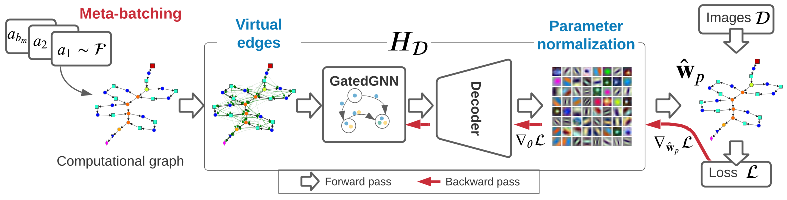
Approaching our task. A straightforward approach to expose $H_{\mathcal{D}}$ to the knowledge of how to optimize other networks is to train it on a large training set of $\{a_i, \mathbf{w}_{p,i}\}$ pairs, however, that is prohibitive[2]. Instead, we follow the bi-level optimization paradigm common in meta-learning [16–18], but rather than iterating over M tasks, we iterate over M training architectures $\mathcal{F} = \{a_i\}_{i=1}^{M}$:

$$\arg\min_{\theta} \sum_{j=1}^{N} \sum_{i=1}^{M} \mathcal{L}\Big(f\Big(\mathbf{x}_j; a_i, H_{\mathcal{D}}(a_i; \theta)\Big), y_j\Big). \tag{2}$$

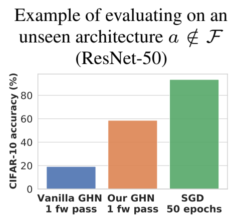
By optimizing Equation 2, the hypernetwork $H_{\mathcal{D}}$ gradually gains knowledge of how to predict performant parameters for training architectures. It can then leverage this knowledge at test time – when predicting parameters for *unseen* architectures. To approach the problem in Equation 2, we need to design the network space $\mathcal{F}$ and $H_{\mathcal{D}}$. For $\mathcal{F}$, we rely on the previous design spaces for neural architectures [19] that we extend in two ways: the ability to sample distinct architectures and an expanded design space that includes diverse architectures, such as ResNets and Visual Transformers [20]. Such architectures can be fully described in the form of computational graphs (Fig. 1). So, to design the hypernetwork $H_{\mathcal{D}}$, we rely on recent advances in machine learning on graph-structured data [21–24]. In particular, we build on the Graph HyperNetworks method (GHNs) [24] that also optimizes Equation 2. However, GHNs do not aim to predict large-scale performant parameters as we do in this work, which motivates us to improve on their approach.

By designing our diverse space $\mathcal{F}$ and improving on GHNs, we boost the accuracy achieved by the predicted parameters on *unseen* architectures to 77% (top-1) and 48% (top-5) on CIFAR-10 [15] and ImageNet [1], respectively. Surprisingly, our GHN shows good out-of-distribution generalization and predicts good parameters for architectures that are much larger and deeper compared to the ones seen in training. For example, we can predict all 24 million parameters of ResNet-50 in less than a second either on a GPU or CPU achieving ∼60% on CIFAR-10 without any gradient updates (Fig 1, (b)).

Overall, our framework and results pave the road toward a new and significantly more efficient paradigm for training networks. Our **contributions** are as follows: (**a**) we introduce the novel task of predicting performant parameters for diverse feedforward neural networks with a single hypernetwork forward pass; (**b**) we introduce DEEPNETS-1M – a standardized benchmark with in-distribution and out-of-distribution architectures to track progress on the task (§ 3); (**c**) we define several baselines and propose a GHN model (§ 4) that performs surprisingly well on CIFAR-10 and ImageNet (§ 5.1); (**d**) we show that our model learns a strong representation of neural network architectures (§ 5.2), and our model is useful for initializing neural networks (§ 5.3). Our DEEPNETS-1M dataset, trained GHNs and code is available at `https://github.com/facebookresearch/ppuda`.



(a)



(b)

Figure 1: **(a)** Overview of our GHN model (§ 4) trained by backpropagation through the predicted parameters ($\hat{\mathbf{w}}_p$) on a given image dataset and our DEEPNETS-1M dataset of architectures. Colored captions show our key improvements to vanilla GHNs (§ 2.2). The red one is used only during training GHNs, while the blue ones are used both at training and testing time. The computational graph of a_1 is visualized as described in Table 1. **(b)** Comparing classification accuracies when all the parameters of a ResNet-50 are predicted by GHNs versus when its parameters are trained with SGD (see full results in § 5).

[2]Training a single network a_i can take several GPU days and thousands of trained networks may be required.

2 Background

We start by providing a brief background about the network design spaces leveraged in the creation of our DEEPNETS-1M dataset of neural architectures described in § 3. We then cover elements of graph hypernetworks that we leverage when designing our specific GHN $H_{\mathcal{D}}$ in § 4.

2.1 Network Design Space of DARTS

DARTS [19] is a differentiable NAS framework. For image classification tasks such as those considered in this work, its networks are defined by four types of building blocks: *stems*, *normal cells*, *reduction cells*, and *classification heads*. Stems are fixed blocks of convolutional operations that process input images. The normal and reduction cells are the main blocks of architectures and are composed of: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity and zero (to indicate the absence of connectivity between two operations). Finally, the classification head defines the network output and is built with a global pooling followed by a single fully connected layer.

Typically, DARTS networks have one stem block, 14-20 cells, and one classification head, altogether forming a deep computational graph. The reduction cells, placed only at 1/3 and 2/3 of the total depth, decrease the spatial resolution and increase the channel dimensionality by a factor of 2. Summation and concatenation are used to aggregate outputs from multiple operations within each cell. To make the channel dimensionalities match, 1×1 convolutions are used as needed. All convolutional operations use the ReLU-Conv-Batch Norm (BN) [7] order. Overall, DARTS enables defining strong architectures that combine many principles of manual [25, 8, 26, 27] and automatic [24, 28–33] design of neural architectures. While DARTS learns the optimal task-specific cells, the framework can be modified to permit sampling randomly-structured cells. We leverage this possibility for the DEEPNETS-1M construction in § 3. Please see § A.1 for further details on DARTS.

2.2 Graph HyperNetwork: GHN-1

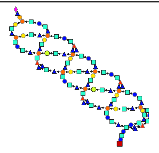
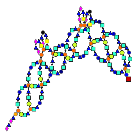
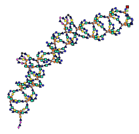
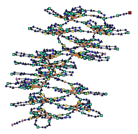
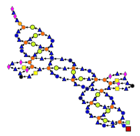
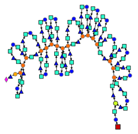
Representation of architectures. GHNs [24] directly operate on the computational graph of a neural architecture a. Specifically, a is a directed acyclic graph (DAG), where nodes $V = \{v_i\}_{i=1}^{|V|}$ are operations (e.g. convolutions, fully-connected layers, summations, etc.) and their connectivity is described by a binary adjacency matrix $\mathbf{A} \in \{0,1\}^{|V|\times|V|}$. Nodes are further characterized by a matrix of initial node features $\mathbf{H}^0 = [\mathbf{h}_1^0, \mathbf{h}_2^0, ..., \mathbf{h}_{|V|}^0]$, where each $\mathbf{h}_v^0$ is a one-hot vector representing the operation performed by the node. We also use such a one-hot representation for $\mathbf{H}^0$, but in addition encode the shape of parameters associated with nodes as described in detail in § B.1.

Design of the graph hypernetwork. In [24], the graph hypernetwork $H_{\mathcal{D}}$ consists of three key modules. The first module takes the input node features $\mathbf{H}^0$ and transforms them into d-dimensional node features $\mathbf{H}^1 \in \mathbb{R}^{|V|\times d}$ through an embedding layer. The second module takes $\mathbf{H}^1$ together with $\mathbf{A}$ and feeds them into a specific variant of the gated graph neural network (GatedGNN) [34]. In particular, their GatedGNN mimics the canonical order π of node execution in the forward (fw) and backward (bw) passes through a computational graph. To do so, it sequentially traverses the graph and performs iterative message passing operations and node feature updates as follows:

$$\forall t \in [1,...,T] : \Big[\forall \pi \in [\text{fw}, \text{bw}] : \Big(\forall v \in \pi : \mathbf{m}_v^t = \sum_{u \in \mathcal{N}_v^\pi} \text{MLP}(\mathbf{h}_u^t), \;\; \mathbf{h}_v^t = \text{GRU}(\mathbf{h}_v^t, \mathbf{m}_v^t)\Big)\Big], \quad (3)$$

where T denotes the total number of forward-backward passes; $\mathbf{h}_v^t$ corresponds to the features of node v in the t-th graph traversal; MLP$(\cdot)$ is a multi-layer perceptron; and GRU$(\cdot)$ is the update function of the Gated Recurrent Unit [35]. In the forward propagation ($\pi = \text{fw}$), $\mathcal{N}_v^\pi$ corresponds to the incoming neighbors of the node defined by $\mathbf{A}$, then in the backward propagation ($\pi = \text{bw}$) it similarly corresponds to the outgoing neighbors of the node. The last module uses the GatedGNN output hidden states $\mathbf{h}_v^T$ to condition a decoder that produces the parameters $\hat{\mathbf{w}}_p^v$ (e.g. convolutional weights) associated with each node. In practice, to handle different parameter dimensionalities per operation type, the output of the hypernetwork is reshaped and sliced according to the shape of parameters in each node. We refer to the model described above as GHN-1 (Fig. 1). Further subtleties of implementing this model in the context of our task are discussed in § B.1.

Table 1: Examples of computational graphs (visualized using NetworkX [44]) in each split and their key statistics, to which we add the average degree and average shortest path length often used to measure local and global graph properties respectively [45, 46]. In the visualized graphs, a node is one of the 15 primitives coded with markers shown at the bottom, where they are sorted by the frequency in the training set. For visualization purposes, a blue triangle marker differentiates a 1×1 convolution (equivalent to a fully-connected layer over channels) from other convolutions, but its primitive type is still just convolution. *Computed based on CIFAR-10.

	In-Distribution		Out-of-Distribution				
							
	Train	Val/Test	Wide	Deep	Dense	BN-free	ResNet/ViT
#graphs	10^6	500/500	100	100	100	100	1/1
#cells	4-18	4-18	4-18	**10-36**	4-18	4-18	16/12
#channels	16-128	32-128	**128-1216**	32-208	32-240	32-336	64/128
#nodes ($\|V\|$)	21-827	33-579	33-579	**74-1017**	**57-993**	33-503	161/114
% w/o BN	3.5%	4.1%	4.1%	2.0%	5.0%	**100%**	0%/**100%**
#params(M)*	0.01-3.1	2.5-35	**39-101**	2.5-15.3	2.5-8.8	2.5-7.7	**23.5**/1.0
avg degree	2.3±0.1	2.3±0.1	2.3±0.1	2.3±0.1	**2.4**±0.1	**2.4**±0.1	2.2/2.3
avg path	14.5±4.8	14.5±4.9	14.7±4.9	**26.2**±9.3	15.1±4.1	10.0±2.8	11.2/10.7

marker	●	■	●	◆	●	▲	●	■	●	●	●	●	■	●	■
primitive	conv	BN	sum	bias	group conv	concat	dilated gr. conv	LN	max pool	avg pool	MSA	SE	input	glob avg	pos enc
fraction in Train (%)	36.3	25.5	11.1	6.5	5.1	3.8	2.5	2.5	1.8	1.7	1.2	1.0	0.5	0.5	0.2

3 DeepNets-1M

The network design space of DARTS is limited by the number of unique operations that compose cells, and the low variety of stems and classification heads. Thus, many architectures are not realizable within this design space, including: VGG [25], ResNets [8], MobileNet [33] or more recent ones such as Visual Transformer (ViT) [20] and Normalization-free networks [36, 37]. Furthermore, DARTS does not define a procedure to sample random architectures. By addressing these two limitations we aim to expose our hypernetwork to diverse training architectures and permit its evaluation on common architectures, such as ResNet-50. We hypothesize that increased training diversity can improve hyper-networks' generalization to unseen architectures making it more competitive to iterative optimizers.

Extending the network design space. We extend the set of possible operations with non-separable 2D convolutions[3], Squeeze&Excite[4] (SE) [40] and Transformer-based operations [41, 20]: multihead self-attention (MSA), positional encoding and layer norm (LN) [42]. Each node (operation) in our graphs has two attributes: *primitive type* (e.g. convolution) and *shape* (e.g. $3\times3\times512\times512$). Overall, our extended set consists of 15 primitive types (Table 1). We also extend the diversity of the generated architectures by introducing VGG-style classification heads and ViT stems. Finally, to further increase architectural diversity, we allow the operations to not include batch norm (BN) [7] and permit networks without channel width expansion (e.g. as in [20]).

Architecture generation process. We generate different subsets of architectures (see the description of each subset in the next two paragraphs and in Table 1). For each subset depending on its purpose, we predefine a range of possible model depths (number of cells), widths and number of nodes per cell. Then, we sample a stem, a normal and reduction cell and a classification head. The internal structure of the normal and reduction cells is defined by uniformly sampling from all available operations. Due to a diverse design space it is extremely unlikely to sample the same architecture multiple times, but we ran a sanity check using the Hungarian algorithm [43] to confirm that (see Figure 6 in § A.2 for details).

In-distribution (ID) architectures. We generate a training set of $|\mathcal{F}| = 10^6$ architectures and validation/test sets of 500/500 architectures that follow the same generation rules and are considered to be ID samples. However, training on large architectures can be prohibitive, e.g. in terms of GPU memory. Thus, in the training set we allow the number of channels and, hence the total number of parameters, to be stochastically defined given computational resources. For example, to train

[3]Non-separable convolutions have weights of e.g. shape $3\times3\times512\times512$ as in ResNet-50. NAS works, such as DARTS and GHN, avoid such convolutions, since the separable ones [38] are more efficient. Non-separable convolutions are nevertheless common in practice and can often boost the downstream performance.

[4]The Squeeze&Excite operation is common in many efficient networks [39, 12].

our models we upper bound the number of parameters in the training architectures to around 3M by sampling fewer channels if necessary. In the evaluation sets, the number of channels is fixed. Therefore, this pre-processing step prior to training results in some distribution shift between the training and the validation/test sets. However, the shift is not imposed by our dataset.

Out-of-distribution (OOD) architectures. We generate five OOD test sets that follow different generation rules. In particular, we define WIDE and DEEP sets that are of interest due the stronger downstream performance of such nets in large-scale tasks [47, 48, 10]. These nets are often more challenging to train for fundamental [49, 50] or computational [51] reasons, so predicting their parameters might ease their subsequent optimization. We also define the DENSE set, since networks with many operations per cell and complex connectivity are underexplored in the literature despite their potential [27]. Next, we define the BN-FREE set that is of interest due to BN's potential negative side-effects [52, 53] and the difficulty or unnecessity of using it in some cases [54–56, 36, 37]. We finally add the RESNET/VIT set with two predefined image classification architectures: commonly-used ResNet-50 [8] and a smaller 12-layer version of the Visual Transformer (ViT) [20] that has recently received a lot of attention in the vision community. Please see § A.1 and § A.2 for further details and statistics of our DEEPNETS-1M dataset.

4 Improved Graph HyperNetworks: GHN-2

In this section, we introduce our three key improvements to the baseline GHN-1 described in § 2.2 (Fig. 1). These components are essential to predict stronger parameters on our task. For the empirical validation of the effectiveness of these components see ablation studies in § 5.1 and § C.2.1.

4.1 Differentiable Normalization of Predicted Parameters

Table 2: Parameter normalizations.

Type of node v	Normalization
Conv./fully-conn.	$\hat{\mathbf{w}}_p^v \sqrt{\beta/(C_{in}\mathcal{H}\mathcal{W})}$
Norm. weights	$2 \times \text{sigmoid}(\hat{\mathbf{w}}_p^v/T)$
Biases	$\tanh(\hat{\mathbf{w}}_p^v/T)$

When training the parameters of a given network from scratch using iterative optimization methods, the initialization of parameters is crucial. A common approach is to use He [57] or Glorot [58] initialization to stabilize the variance of activations across layers of the network. Chang et al. [59] showed that when the parameters of the network are instead predicted by a hypernetwork, the activations in the network tend to explode or vanish. To address the issue of unstable network activations especially for the case of predicting parameters of diverse architectures, we apply *operation-dependent normalizations* (Table 2). We normalize convolutional and fully-connected weights by following the *fan-in* scheme of [57] (see the comparison to *fan-out* in § C.2.1): $\hat{\mathbf{w}}_p^v \sqrt{\beta/(C_{in}\mathcal{H}\mathcal{W})}$, where $C_{in}, \mathcal{H}, \mathcal{W}$ are the number of input channels and spatial dimensions of weights $\hat{\mathbf{w}}_p^v$, respectively; and β is a nonlinearity specific constant following the analysis in [57]. The parameters of normalization layers such as BN and LN, as well as biases typically initialized with constants, are normalized by applying a squashing function with temperature T to imitate the empirical distributions of models trained with SGD (see Table 2). These are differentiable normalizations, so that they are applied at training (and testing) time. Further analysis of our normalization and its stabilizing effect on activations is presented in § B.2.2.

4.2 Enhancing Long-range Message Propagation

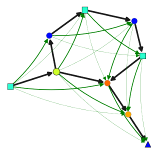


Figure 2: Virtual edges (in green) allow for better capture of global context.

Computational graphs often take the form of long chains (Table 1) with only a few incoming/outcoming edges per node. This structure might hinder long-range propagation of information between nodes [60]. Different approaches to alleviate the long-range propagation problem exist [61–63], including stacking GHNs in [24]. Instead we adopt simple graph-based heuristics in line with recent works [64, 65]. In particular, we add *virtual edges* between two nodes v and u and weight them based on the shortest path s_{vu} between them (Fig. 2). To avoid interference with the *real* edges in the computational graph, we introduce a separate MLP$_{\text{sp}}$ to transform the features of the nodes connected through these virtual edges, and redefine the message passing of Equation 3 as:

$$\mathbf{m}_v^t = \sum\nolimits_{u \in \mathcal{N}_v^{\pi}} \text{MLP}(\mathbf{h}_u^t) + \sum\nolimits_{u \in \mathcal{N}_v^{(\text{sp})}} \frac{1}{s_{vu}} \text{MLP}_{\text{sp}}(\mathbf{h}_u^t), \tag{4}$$

where $\mathcal{N}_v^{(sp)}$ are neighbors satisfying $1 < s_{vu} \leq s^{(\max)}$, and $s^{(\max)}$ is a hyperparameter. To maintain the same number of trainable parameters as in GHN-1, we decrease MLPs' sizes appropriately. Despite its simplicity, this approach is effective (see the comparison to stacking GHNs in § C.2.1).

4.3 Meta-batching Architectures During Training

GHN-1 updates its parameters θ based on a single architecture sampled for each batch of images (Equation 2). In vanilla SGD training, larger batches of images often speed up convergence by reducing gradient noise and improve model's performance [66]. Therefore, we define a meta-batch b_m as the number of architectures sampled per batch of images. Both the parameter prediction and the forward/backward passes through the architectures in a meta-batch can be done in parallel. We then average the gradients across b_m to update the parameters θ of $H_{\mathcal{D}}$: $\nabla_\theta \mathcal{L} = 1/b_m \sum_{i=1}^{b_m} \nabla_\theta \mathcal{L}_i$. Further analysis of the meta-batching effect on the training loss and convergence speed is presented in § B.2.3.

5 Experiments

We focus the evaluation of GHN-2 on our parameter prediction task (§ 5.1). In addition, we show beneficial side-effects of i) learning a stronger neural architecture representation using GHN-2 in analyzing networks (§ 5.2) and ii) predicting parameters for fine-tuning (§ 5.3). We provide further experimental and implementation details, as well as more results supporting our arguments in § C.

Datasets. We use the DeepNets-1M dataset of architectures (§ 3) as well as two image classification datasets $\mathcal{D}_1$ (CIFAR-10 [15]) and $\mathcal{D}_2$ (ImageNet [1]). CIFAR-10 consists of 50k training and 10k test images of size 32×32×3 and 10 object categories. ImageNet is a larger scale dataset with 1.28M training and 50k test images of variable size and 1000 fine-grained object categories. We resize ImageNet images to 224×224×3 following [19, 24]. We use 5k/50k training images as a validation set in CIFAR-10/ImageNet and 500 validation architectures of DeepNets-1M for hyperparameter tuning.

Baselines. Our baselines include GHN-1 and a simple MLP that only has access to operations, but not to the connections between them. This MLP baseline is obtained by replacing the GatedGNN with an MLP in our GHN-2. Since GHNs were originally introduced for small architectures of ~ 50 nodes and only trained on CIFAR-10, we reimplement[5] them and scale them up by introducing minor modifications to their decoder that enable their training on ImageNet and on larger architectures of up to 1000 nodes (see § B.1 for details). We use the same hyperparameters to train the baselines and GHN-2.

Iterative optimizers. In the parameter prediction experiments, we also compare our model to standard optimization methods: SGD and Adam [3]. We use off-the-shelf hyperparameters common in the literature [24, 19, 32, 67–69]. On CIFAR-10, we train evaluation architectures with SGD/Adam, initial learning rate $\eta = 0.025$ / $\eta = 0.001$, batch size $b = 96$ and up to 50 epochs. With Adam, we train only 300 evaluation architectures as a rough estimation of an average performance. On ImageNet, we train them with SGD, $\eta = 0.1$ and $b = 128$, and, for computational reasons (given 1402 evaluation architectures in total), we limit training with SGD to 1 epoch. We have also considered meta-optimizers, such as [17, 18]. However, we were unable to scale them to diverse and large architectures of our DeepNets-1M, since their LSTM requires a separate hidden state for every trainable parameter in the architecture. The scalable variants exist [70, 71], but are hard to reproduce without open source code.

Additional experimental details. We follow [24] and train GHNs with Adam, $\eta = 0.001$ and batch size of 64 images for CIFAR-10 and 256 for ImageNet. We train for up to 300 epochs, except for one experiment in the ablation studies, where we train one GHN with $b_m = 1$ eight times longer, i.e. for 2400 epochs. All GHNs in our experiments use $T = 1$ propagation (Equation 3), as we found the original $T = 5$ of [24] to be inefficient and it did not improve the accuracies in our task. GHN-2 uses $s^{(\max)} = 50$ and $b_m = 8$ and additionally uses LN that slightly further improves results (see these ablations in § C.2.1). Model selection is performed on the validation sets, but the results in our paper are reported on the test sets to enable their direct comparison.

5.1 Parameter Prediction

Experimental setup. We trained our GHN-2 and baselines on the training architectures and training images, i.e. a separate model is trained for CIFAR-10 and ImageNet. According to our DeepNets-1M benchmark, we assess whether these models can generalize to unseen in-distribution (ID) and out-of-distribution (OOD) test architectures from our DeepNets-1M. We measure this generalization by predicting parameters for the test architectures and computing their classification accuracies on the test images of CIFAR-10 (Table 3) and ImageNet (Table 4). The evaluation architectures with batch norm (BN) have running statistics, which are not learned by gradient descent [7], and

[5]While source code for GHNs [24] is unavailable, we appreciate the authors' help in implementing some steps.

hence are not predicted by our GHNs. To alleviate that, we follow [24] and evaluate the networks with BN by computing per batch statistics with batch size of 64 images. This is further discussed in § C.1.

Results. Despite GHN-2 never observed the test architectures, GHN-2 predicts good parameters for them making the test networks perform surprisingly well on both image datasets (Tables 3 and 4). Our results are especially strong on CIFAR-10, where some architectures with predicted parameters achieve up to 77.1%, while the best accuracy of training with SGD for 50 epochs is around 15% more. We even show good results on ImageNet, where for some architectures we achieve a top-5 accuracy of up to 48.3%. While these results are low for direct downstream applications, they are remarkable for three main reasons. First, to train GHNs by optimizing Equation 2, we do not rely on the prohibitively expensive procedure of training the architectures $\mathcal{F}$ by SGD. Second, GHNs rely on a single forward pass to predict all parameters. Third, these results are obtained for unseen architectures, including the OOD ones. Even in the case of severe distribution shifts (e.g. ResNet-50[6]) and underrepresented networks (e.g. ViT[7]), our model still predicts parameters that perform better than random ones. On CIFAR-10, generalization of GHN-2 is particularly strong with a 58.6% accuracy on ResNet-50.

On both image datasets, our GHN-2 significantly outperforms GHN-1 on all test subsets of DeepNets-1M with more than a 20% absolute gain in certain cases, e.g. 36.8% vs 13.7% on the BN-free networks (Table 3). Exploiting the structure of computational graphs is a critical property of GHNs with the accuracy dropping from 66.9% to 42.2% on ID (and even more on OOD) architectures when we replace the GatedGNN of GHN-2 with an MLP. Compared to iterative optimization methods, GHN-2 predicts parameters achieving an accuracy similar to $\sim$2500 and $\sim$5000 iterations of SGD on CIFAR-10 and ImageNet respectively. In contrast, GHN-1 performs similarly to only $\sim$500 and $\sim$2000 (not shown in Table 4) iterations respectively. Comparing SGD to Adam, the latter performs worse in general except for the ViT architectures similar to [72, 20].

To report speeds on ImageNet in Table 4, we use a dedicated machine with a single NVIDIA V100-32GB and Intel Xeon CPU E5-1620 v4@ 3.50GHz. So for SGD these numbers can be reduced by using faster computing infrastructure and more optimal hyperparameters [73]. Using our setup,

Table 3: CIFAR-10 results of predicted parameters for unseen ID and OOD architectures of DeepNets-1M. Mean ($\pm$standard error of the mean) accuracies are reported (random chance $\approx$10%). †The number of parameter updates.

Method	#upd†	ID-test		OOD-test				
		avg	max	Wide	Deep	Dense	BN-free	ResNet/ViT
MLP	1	42.2±0.6	60.2	22.3±0.9	37.9±1.2	44.8±1.1	23.9±0.7	17.7/10.0
GHN-1	1	51.4±0.4	59.9	43.1±1.7	48.3±0.8	51.8±0.9	13.7±0.3	19.2/**18.2**
GHN-2	1	**66.9**±0.3	**77.1**	**64.0**±1.1	**60.5**±1.2	**65.8**±0.7	**36.8**±1.5	**58.6**/11.4
Iterative optimizers (all architectures are ID in this case)								
SGD (1 epoch)	0.5×10^3	46.1±0.4	66.5	47.2±1.1	34.2±1.1	45.3±0.7	18.0±1.1	61.8/34.5
SGD (5 epochs)	2.5×10^3	69.2±0.4	82.4	71.2±0.3	56.7±1.6	67.8±0.9	29.0±2.0	78.2/52.5
SGD (50 epochs)	25×10^3	88.5±0.3	93.1	88.9±1.2	84.5±1.2	87.3±0.8	45.6±3.6	93.5/75.7
Adam (50 epochs)	25×10^3	84.0±0.8	89.5	82.0±1.6	76.2±2.6	84.8±0.4	38.8±4.8	91.5/79.4

Table 4: ImageNet results on DeepNets-1M. Mean ($\pm$standard error of the mean) top-5 accuracies are reported (random chance $\approx$0.5%). *Estimated on ResNet-50 with batch size 128.

Method	#upd	GPU sec.	CPU sec.	ID-test		OOD-test				
		avg	avg	avg	max	Wide	Deep	Dense	BN-free	ResNet/ViT
GHN-1	1	0.3	0.5	17.2±0.4	32.1	15.8±0.9	15.9±0.8	15.1±0.7	0.5±0.0	**6.9**/0.9
GHN-2	1	0.3	0.7	**27.2**±0.6	**48.3**	**19.4**±1.4	**24.7**±1.4	**26.4**±1.2	**7.2**±0.6	5.3/**4.4**
Iterative optimizers (all architectures are ID in this case)										
SGD (1 step)	1	0.4	6.0	0.5±0.0	0.7	0.5±0.0	0.5±0.0	0.5±0.0	0.5±0.0	0.5/0.5
SGD (5000 steps)	5k	2×10^3	3×10^4	25.6±0.3	50.7	26.2±1.4	13.2±1.1	25.4±1.1	4.8±0.8	43.6/24.3
SGD (10000 steps)	10k	4×10^3	6×10^4	37.7±0.6	62.0	38.7±1.6	22.1±1.4	36.3±1.2	8.0±1.2	57.7/33.4
SGD (100 epochs)	1000k	$6\times10^{5*}$	$6\times10^{7*}$	–		–	–	–	–	92.9/72.4

[6]Large architectures with bottleneck layers such as ResNet-50 do not appear during training.

[7]Architectures such as ViT do not include BN and, except for the first layer, convolutions – the two most frequent operations in the training set.

SGD requires on average $10^4\times$ more time on a GPU ($10^5\times$ on a CPU) to obtain parameters that yield performance similar to GHN-2. As a concrete example, AlexNet [74] requires around 50 GPU hours (on our setup) to achieve a 81.8% top-5 accuracy, while on some architectures we achieve $\geq$48.0% in just 0.3 GPU seconds.

Table 5: Ablating GHN-2 on CIFAR-10. An average rank of the model is computed across all ID and OOD test architectures.

MODEL	ID-TEST	OOD-TEST	AVG. RANK
GHN-2	**66.9**$\pm$0.3	**56.8**$\pm$0.8	**1.9**
1000 training architectures	65.1$\pm$0.5	52.5$\pm$1.0	2.6
No normalization (§ 4.1)	62.6$\pm$0.6	47.1$\pm$1.2	3.9
No virtual edges (§ 4.2)	61.5$\pm$0.4	53.9$\pm$0.6	4.1
No meta-batch ($b_m = 1$, § 4.3)	54.3$\pm$0.3	47.5$\pm$0.6	5.5
$b_m = 1$, train $8\times$ longer	62.4$\pm$0.5	51.9$\pm$1.0	3.7
No GatedGNN (MLP)	42.2$\pm$0.6	32.2$\pm$0.7	7.4
GHN-1	51.4$\pm$0.4	39.2$\pm$0.9	6.8

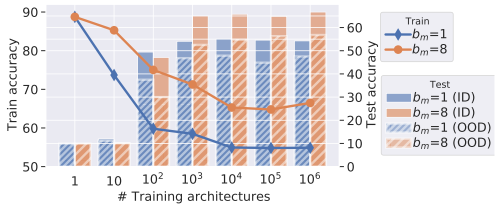


Figure 3: GHN-2 with meta batch $b_m = 8$ versus $b_m = 1$ for different numbers of training architectures on CIFAR-10.

Ablations (Table 5) show that all three components proposed in § 4 are important. Normalization is particularly important for OOD generalization with the largest drops on the WIDE and BN-FREE networks (see § C.2.1). Using meta-batching ($b_m = 8$) is also essential and helps stabilize training and accelerate convergence (see § B.2). We also confirm that the performance gap between $b_m = 1$ and $b_m = 8$ is not primarily due to the observation of more architectures, since the ablated GHN-2 with $b_m = 1$ trained eight times longer is still inferior. The gap between $b_m = 8$ and $b_m = 1$ becomes pronounced with *at least* 1k training architectures (Fig. 3). When training with fewer architectures (e.g. 100), the GHN with meta-batching starts to overfit to the training architectures. Given our challenging setup with unseen evaluation architectures, it is surprising that using 1k training architectures already gives strong results. However, OOD generalization degrades in this case compared to using all 1M architectures, especially on the BN-FREE networks (see § B.2). When training GHNs on just a few architectures, the training accuracy soars to the level of training them with SGD. With more architectures, it generally decreases indicating classic overfitting and underfitting cases.

5.2 Property Prediction

Representing computational graphs of neural architectures is a challenging problem [75–79]. We verify if GHNs are capable of doing that out-of-the-box in the property prediction experiments. We also experiment with architecture comparison in § C.2.4. Our hypothesis is that by better solving our parameter prediction task, GHNs should also better solve graph representation tasks.

Experimental setup. We predict the properties of architectures given their graph embeddings obtained by averaging node features[8]. We consider four such properties (see § C.2.3 for details):

- Accuracy on the "clean" (original) validation set of images;
- Accuracy on a corrupted set (obtained by adding the Gaussian noise to images following [53]);
- Inference speed (latency or GPU seconds per a batch of images);
- Convergence speed (the number of SGD iterations to achieve a certain training accuracy).

Estimating these properties accurately can have direct practical benefits. Clean and corrupted accuracies can be used to search for the best performing architectures (e.g. for the NAS task); inference speed can be used to choose the fastest network, so by estimating these properties we can trade-off accurate, robust and fast networks [12]. Convergence speed can be used to find networks that are easier to optimize. These properties correlate poorly with each other and between CIFAR-10 and ImageNet (§ C.2.3), so they require the model to capture different regularities of graphs. While specialized methods to estimate some of these properties exist, often as a NAS task [80–82, 30, 75], our GHNs provide a generic representation that can be easily used for many such properties. For each property, we train a simple regression model using graph embeddings and ground truth property values. We use 500 validation architectures of DEEPNETS-1M for training the regression model and tuning its hyperparameters (see § C.2.3 for details). We then use 500 testing architectures of DEEPNETS-1M to measure Kendall's Tau rank correlation between the predicted and ground truth property values similar to [80].

[8]A fixed size graph embedding for the architecture a can be computed by averaging the output node features: $\mathbf{h}_a = \frac{1}{|V|}\sum_{v\in V}\mathbf{h}_v^T$, where $\mathbf{h}_a \in \mathbb{R}^d$ and d is the dimensionality of node features.

Additional baseline. We compare to the Neural Predictor (NeuPred) [80]. NeuPred is based on directed graph convolution and is developed for accuracy prediction achieving strong NAS results. We train a separate such NeuPred for each property from scratch following their hyperparameters.

Results. GHN-2 consistently outperforms the GHN-1 and MLP baselines as well as NeuPred (Fig. 4). In § C.2.3, we also provide results verifying if higher correlations translate to downstream gains. For example, on CIFAR-10 by choosing the most accurate architecture according to the regression model and training it from scratch following [19, 24], we obtained a 97.26%($\pm$0.09) accuracy, which is competitive with leading NAS approaches, e.g. [19, 24, 32, 67–69]. In contrast, the network chosen by the regression model trained on the GHN-1 embeddings achieves 95.90%($\pm$0.08).

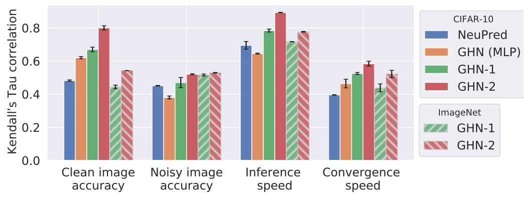


Figure 4: Property prediction of neural networks in terms of correlation (higher is better). Error bars denote the standard deviation across 5 runs.

5.3 Fine-tuning Predicted Parameters

Neural networks trained on ImageNet and other large datasets have proven useful in diverse visual tasks in the transfer learning setup [83–87, 20]. Therefore, we explore how predicting parameters on ImageNet with GHNs compares to pretraining them on ImageNet with SGD in such a setup. We consider low-data tasks as they often benefit more from transfer learning [86, 87].

Experimental setup. We perform two transfer-learning experiments. The first experiment is fine-tuning the predicted parameters on 1,000 training samples (100 labels per class) of CIFAR-10. We fine-tune ResNet-50, Visual Transformer (ViT) and a 14-cell architecture based on the DARTS best cell [19]. The hyperparameters of fine-tuning (initial learning rate and weight decay) are tuned on 200 validation samples held-out of the 1,000 training samples. The number of epochs is fixed to 50 as in § 5.1 for simplicity. In the second experiment, we fine-tune the predicted parameters on the object detection task. We closely follow the experimental protocol and hyperparameters from [88] and train the networks on the Penn-Fudan dataset [89]. The dataset contains only 170 images and the task is to detect pedestrians. Therefore this task is also well suited for transfer learning. Following [88], we replace the backbone of a Faster R-CNN with one of the three architectures. To perform transfer learning with GHNs, in both experiments we predict the parameters of a given architecture using GHNs trained on ImageNet. We then replace the ImageNet classification layer with the target task-specific layers and fine-tune the entire network on the target task. We compare the results of GHNs to He's initialization [57] and the initialization based on pretraining the parameters on ImageNet with SGD.

Table 6: CIFAR-10 test set accuracies and Penn-Fudan object detection average precision (at IoU=0.50) after fine-tuning the networks using SGD initialized with different methods. Average results and standard deviations for 3 runs with different random seeds are shown. For each architecture, similar GHN-2-based and ImageNet-based results are bolded.[*]Estimated on ResNet-50.

INITIALIZATION METHOD	GPU sec. to init.[*]	100-SHOT CIFAR-10			PENN-FUDAN OBJECT DETECTION		
		RESNET-50	VIT	DARTS	RESNET-50	VIT	DARTS
He's [57]	0.003	41.0$\pm$0.4	33.2$\pm$0.3	45.4$\pm$0.4	0.197$\pm$0.042	0.144$\pm$0.010	0.486$\pm$0.035
GHN-1 (trained on ImageNet)	0.6	46.6$\pm$0.0	23.3$\pm$0.1	49.2$\pm$0.1	0.433$\pm$0.013	0.0$\pm$0.0	0.468$\pm$0.024
GHN-2 (trained on ImageNet)	0.7	**56.4**$\pm$0.1	**41.4**$\pm$0.6	**60.7**$\pm$0.3	**0.560**$\pm$0.019	**0.436**$\pm$0.032	**0.785**$\pm$0.032
ImageNet (1k pretraining steps)	6×10^2	45.4$\pm$0.3	**44.3**$\pm$0.1	**62.4**$\pm$0.3	0.302$\pm$0.022	0.182$\pm$0.046	**0.814**$\pm$0.033
ImageNet (2.5k pretraining steps)	1.5×10^3	**55.4**$\pm$0.2	50.4$\pm$0.3	70.4$\pm$0.2	**0.571**$\pm$0.056	0.322$\pm$0.073	0.823$\pm$0.022
ImageNet (5 pretraining epochs)	3×10^4	84.6$\pm$0.2	70.2$\pm$0.5	83.9$\pm$0.1	0.723$\pm$0.045	0.391$\pm$0.024	0.827$\pm$0.053
ImageNet (final epoch)	6×10^5	89.2$\pm$0.2	74.5$\pm$0.2	85.6$\pm$0.2	0.876$\pm$0.011	**0.468**$\pm$0.023	0.881$\pm$0.023

Results. The CIFAR-10 image classification results of fine-tuning the parameters predicted by our GHN-2 are $\geq$10 percentage points better (in absolute terms) than fine-tuning the parameters predicted by GHN-1 or training the parameters initialized using He's method (Table 6). Similarly, the object detection results of GHN-2-based initialization are consistently better than both GHN-1 and He's initializations. The GHN-2 results are a factor of 1.5-3 improvement over He's for all the three architectures. Overall, the two experiments clearly demonstrate the practical value of predicting parameters using our GHN-2. Using GHN-1 for initialization provides relatively small gains or hurts convergence (for ViT). Compared to pretraining on ImageNet with SGD, initialization using GHN-2 leads to performance similar to 1k-2.5k steps of pretraining on ImageNet depending on the architecture in the case of CIFAR-10. In the case of Penn-Fudan, GHN-2's performance is similar to $\geq$1k steps of pretraining with SGD. In both experiments, pretraining on ImageNet for just 5 epochs provides strong transfer learning performance and the final ImageNet checkpoints are only slightly better, which aligns with previous works [85]. Therefore, further improvements in the parameter prediction models appear promising.

6 Related Work

Our proposed parameter prediction task, objective in Equation 2 and improved GHN are related to a wide range of machine learning frameworks, in particular meta-learning and neural architecture search (NAS). Meta-learning is a general framework [16, 90] that includes meta-optimizers and meta-models, among others. Related NAS works include differentiable [19] and one-shot methods [12]. See additional related work in § D.

Meta-optimizers. Meta-optimizers [17, 18, 71, 91, 92] define a problem similar to our task, but where $H_{\mathcal{D}}$ is an RNN-based model predicting the gradients $\nabla \mathbf{w}$, mimicking the behavior of iterative optimizers. Therefore, the objective of meta-optimizers may be phrased as *learning to optimize* as opposed to our *learning to predict* parameters. Such meta-optimizers can have their own hyperparameters that need to be tuned for a given architecture a and need to be run expensively (on the GPU) for many iterations following Equation 1.

Meta-models. Meta-models include methods based on MAML [93], ProtoNets [94] and auxiliary nets predicting task-specific parameters [95–98]. These methods are tied to a particular architecture and need to be trained from scratch if it is changed. Several recent methods attempt to relax the choice of architecture in meta-learning. T-NAS [99] combines MAML with DARTS [19] to learn both the optimal architecture and its parameters for a given task. However, the best network, a, needs to be trained using MAML from scratch. Meta-NAS [100] takes a step further and only requires fine-tuning of a on a given task. However, the a is obtained from a single meta-architecture and so its choice is limited, preventing parameter prediction for arbitrary a. CATCH [101] follows a similar idea, but uses reinforcement learning to quickly search for the best a on the specific task. Overall meta-learning mainly aims at generalization *across tasks*, often motivated by the few-shot learning problem. In contrast, our parameter prediction problem assumes a single task (here an image dataset), but aims at generalization *across architectures* a with the ability to predict parameters in a single forward pass.

One-shot NAS. One-shot NAS aims to learn a single "supernet" [102, 12, 103] that can be used to estimate the performance of smaller nets (subnets) obtained by some kind of pruning the supernet, followed by training the best chosen a from scratch with SGD. Recent models, in particular BigNAS [12] and OnceForAll (OFA) [102], eliminate the need to train subnets. However, the fundamental limitation of one-shot NAS is poor scaling with the number of possible computational operations [24]. This limits the diversity of architectures for which parameters can be obtained. For example, all subnets in OFA are based on MobileNet-v3 [33], which does not allow to solve our more general parameter prediction task. To mitigate this, SMASH [104] proposed to predict some of the parameters using hypernetworks [14] by encoding architectures as a 3D tensor. Graph HyperNetworks (GHNs) [24] further generalized this approach to "arbitrary" computational graphs (DAGs), which allowed them to improve NAS results. GHNs focused on obtaining reliable subnetwork rankings for NAS and did not aim to predict large-scale performant parameters. We show that the vanilla GHNs perform poorly on our parameter prediction task mainly due to the inappropriate scale of predicted parameters, lack of long-range interactions in the graphs, gradient noise and slow convergence when optimizing Equation 2. Conventionally to NAS, GHNs were also trained in a quite constrained architecture space [105]. We expand the architecture space adopting GHNs for a more general problem.

7 Conclusion

We propose a novel framework and benchmark to learn and evaluate neural parameter prediction models. Our model (GHN-2) is able to predict parameters for very diverse and large-scale architectures in a single forward pass in a fraction of a second. The networks with predicted parameters yield surprisingly high image classification accuracy given the extremely challenging nature of our parameter prediction task. However, the accuracy is still far from networks trained with handcrafted optimization methods. Bridging the gap is a promising future direction. As a beneficial side-effect, GHN-2 learns a strong representation of neural architectures as evidenced by our property prediction evaluation. Finally, parameters predicted using GHN-2 trained on ImageNet benefit transfer learning in the low-data regime. This motivates further research towards solving our task.

Acknowledgments

BK is thankful to Facebook AI Research for funding the initial phase of this research during his internship and to NSERC and the Ontario Graduate Scholarship used to fund the other phases of this research. GWT and BK also acknowledge support from CIFAR and the Canada Foundation for Innovation. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute: `http://www.vectorinstitute.ai/#partners`. We are thankful to Magdalena Sobol for editorial help. We are thankful to the Vector AI Engineering team (Gerald Shen, Maria Koshkina and Deval Pandya) for code review. We are also thankful to the reviewers for their constructive feedback.

References

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[2] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[5] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Freezeout: Accelerate training by progressively freezing layers. *arXiv preprint arXiv:1706.04983*, 2017.

[6] Dami Choi, Alexandre Passos, Christopher J Shallue, and George E Dahl. Faster neural network training with data echoing. *arXiv preprint arXiv:1907.05550*, 2019.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] NVIDIA. Nvidia data center deep learning product performance. URL `https://developer.nvidia.com/deep-learning-performance-training-inference`.

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[11] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[12] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment, 2019.

[13] Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020.

[14] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[15] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[16] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[17] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[18] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[21] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

[22] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

[23] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[24] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[27] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

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

[29] 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*, pages 8697–8710, 2018.

[30] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

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

[32] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive darts: Bridging the optimization gap for nas in the wild. *arXiv preprint arXiv:1912.10952*, 2019.

[33] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[34] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[35] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[36] Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. *arXiv preprint arXiv:2101.08692*, 2021.

[37] Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

[38] Laurent Sifre and Stéphane Mallat. Rigid-motion scattering for texture classification. *arXiv preprint arXiv:1403.1687*, 2014.

[39] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[40] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[42] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[43] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[44] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[45] Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. The architecture of complex weighted networks. *Proceedings of the national academy of sciences*, 101(11):3747–3752, 2004.

[46] Jiaxuan You, Jure Leskovec, Kaiming He, and Saining Xie. Graph structure of neural networks, 2020.

[47] Anna Golubeva, Behnam Neyshabur, and Guy Gur-Ari. Are wider nets better given the same number of parameters? *arXiv preprint arXiv:2010.14495*, 2020.

[48] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[49] Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks. In *International conference on machine learning*, pages 2603–2612. PMLR, 2017.

[50] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. *arXiv preprint arXiv:1507.06228*, 2015.

[51] Sara Hooker. The hardware lottery, 2020.

[52] Angus Galloway, Anna Golubeva, Thomas Tanay, Medhat Moussa, and Graham W Taylor. Batch normalization is a cause of adversarial vulnerability. *arXiv preprint arXiv:1905.02161*, 2019.

[53] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[54] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[55] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

[56] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[58] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[59] Oscar Chang, Lampros Flokas, and Hod Lipson. Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*, 2019.

[60] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

[61] Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499, 1996.

[62] Meng Liu, Zhengyang Wang, and Shuiwang Ji. Non-local graph neural networks. *arXiv preprint arXiv:2005.14612*, 2020.

[63] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

[64] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *International Conference on Machine Learning*, pages 7134–7143. PMLR, 2019.

[65] Yiding Yang, Xinchao Wang, Mingli Song, Junsong Yuan, and Dacheng Tao. Spagan: Shortest path graph attention network. *arXiv preprint arXiv:2101.03464*, 2021.

[66] Pavlo M Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology and Management Science*, 20(1):20–24, 2017.

[67] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1829–1838, 2020.

[68] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11993–12002, 2020.

[69] Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1620–1630, 2020.

[70] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, pages 3751–3760. PMLR, 2017.

[71] Luke Metz, Niru Maheswaranathan, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020.

[72] Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J Reddi, Sanjiv Kumar, and Suvrit Sra. Why adam beats sgd for attention models. *arXiv e-prints*, pages arXiv–1912, 2019.

[73] 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.

[74] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[75] Wei Li, Shaogang Gong, and Xiatian Zhu. Neural graph embedding for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4707–4714, 2020.

[76] Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. *arXiv preprint arXiv:1912.00848*, 2019.

[77] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.

[78] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[79] Ilya Makarov, Dmitrii Kiselev, Nikita Nikitinsky, and Lovro Subelj. Survey on graph embeddings and their applications to machine learning problems on graphs. *PeerJ Computer Science*, 7, 2021.

[80] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pages 660–676. Springer, 2020.

[81] Jovita Lukasik, David Friede, Heiner Stuckenschmidt, and Margret Keuper. Neural architecture performance prediction using graph neural networks. *arXiv preprint arXiv:2010.10024*, 2020.

[82] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[83] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2661–2671, 2019.

[84] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[85] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *arXiv preprint arXiv:2008.11687*, 2020.

[86] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. *arXiv preprint arXiv:1902.07208*, 2019.

[87] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[88] PyTorch. Pytorch object detection finetuning tutorial. URL `https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html`.

[89] Liming Wang, Jianbo Shi, Gang Song, and I-fan Shen. Object detection combining recognition and segmentation. In *Asian conference on computer vision*, pages 189–199. Springer, 2007.

[90] Jürgen Schmidhuber and AI Blog. Metalearning machines learn to learn (1987-).

[91] Louis Kirsch and Jürgen Schmidhuber. Meta learning backpropagation and improving it. *arXiv preprint arXiv:2012.14905*, 2020.

[92] Hugo Siqueira Gomes, Benjamin Léger, and Christian Gagné. Meta learning black-box population-based optimizers. *arXiv preprint arXiv:2103.03526*, 2021.

[93] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[94] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.

[95] Adriana Romero, Pierre Luc Carrier, Akram Erraqabi, Tristan Sylvain, Alex Auvolat, Etienne Dejoie, Marc-André Legault, Marie-Pierre Dubé, Julie G Hussin, and Yoshua Bengio. Diet networks: thin parameters for fat genomics. *arXiv preprint arXiv:1611.09340*, 2016.

[96] James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E Turner. Fast and flexible multi-task classification using conditional neural adaptive processes. *arXiv preprint arXiv:1906.07697*, 2019.

[97] Huaiyu Li, Weiming Dong, Xing Mei, Chongyang Ma, Feiyue Huang, and Bao-Gang Hu. Lgm-net: Learning to generate matching networks for few-shot learning. In *International conference on machine learning*, pages 3825–3834. PMLR, 2019.

[98] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip HS Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. *arXiv preprint arXiv:1606.05233*, 2016.

[99] Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta learning. In *ICLR*. JMLR. org, 2020.

[100] Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12365–12375, 2020.

[101] Xin Chen, Yawen Duan, Zewei Chen, Hang Xu, Zihao Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. Catch: Context-based meta reinforcement learning for transferrable architecture search. In *European Conference on Computer Vision*, pages 185–202. Springer, 2020.

[102] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. *arXiv preprint arXiv:2003.11142*, 2020.

[103] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[104] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[105] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 550–559, 2018.

[106] Guixiang Ma, Nesreen K Ahmed, Theodore L Willke, and S Yu Philip. Deep graph similarity learning: A survey. *Data Mining and Knowledge Discovery*, pages 1–38, 2021.

[107] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 384–392, 2019.

[108] DC Dowson and BV Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

[109] Chia-Cheng Liu, Harris Chan, Kevin Luk, and AI Borealis. Auto-regressive graph generation modeling with improved evaluation methods. In *33rd Conference on Neural Information Processing Systems. Vancouver, Canada*, 2019.

[110] Julian Zilly, Hannes Zilly, Oliver Richter, Roger Wattenhofer, Andrea Censi, and Emilio Frazzoli. The frechet distance of training and test distribution predicts the generalization gap. 2019.

[111] Rylee Thompson, Elahe Ghalebi, Terrance DeVries, and Graham W Taylor. Building lego using deep generative models of graphs. *arXiv preprint arXiv:2012.11543*, 2020.

[112] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[113] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In C J C Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2148–2156. Curran Associates, Inc., 2013.

[114] Neale Ratzlaff and Li Fuxin. Hypergan: A generative model for diverse, performant neural networks. In *International Conference on Machine Learning*, pages 5361–5369. PMLR, 2019.

[115] Iou-Jen Liu, Jian Peng, and Alexander G Schwing. Knowledge flow: Improve upon your teachers. *arXiv preprint arXiv:1904.05878*, 2019.

[116] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[117] Yann Dauphin and Samuel S Schoenholz. Metainit: Initializing learning by learning to initialize. 2019.

[118] Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W Ronny Huang, and Tom Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *arXiv preprint arXiv:2102.08098*, 2021.

[119] Debasmit Das, Yash Bhalgat, and Fatih Porikli. Data-driven weight initialization with sylvester solvers. *arXiv preprint arXiv:2105.10335*, 2021.

[120] Yue Yu, Jie Chen, Tian Gao, and Mo Yu. Dag-gnn: Dag structure learning with graph neural networks. In *International Conference on Machine Learning*, pages 7154–7163. PMLR, 2019.

[121] Xiaojie Guo and Liang Zhao. A systematic survey on deep generative models for graph generation. *arXiv preprint arXiv:2007.06686*, 2020.

[122] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

[123] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

Appendix

Table of Contents

A DEEPNETS-1M Details

A.1 Generating DEEPNETS-1M using DARTS

In this section, we elaborate on our description in § 2.1 about how DARTS [19] defines networks. We also elaborate on our discussion in § 3 about how we modify the DARTS framework to generate our DEEPNETS-1M dataset of architectures and summarize these modifications here, in Table 7. We visualize examples of architectures defined using DARTS [19] and corresponding computational graphs obtained using our code (Fig. 5).

Overall architecture structure. At a high level, all our ID and OOD networks are composed of a stem, repeated normal and reduction cells, global average pooling and a classification head (Fig. 5, (a)). We optionally sample fully-connected layers between the global pooling and the last classification layer and/or replace global pooling with fully connected layers, e.g. as in VGG [25]. The stem in DARTS, and in other NAS works, is predefined and fixed for each image dataset. We uniformly sample either a CIFAR-10 style or ImageNet style stem, so that our network space is unified for both image datasets. To prevent extreme GPU memory consumption when using a non-ImageNet stem for ImageNet images, we additionally use a larger stride in the stem that does not affect the graph structure. **At test time**, however, we can predict parameters for networks without these constraints, but the performance of the predicted parameters might degrade accordingly. For example, we can successfully predict parameters for ResNet-50 (Fig. 5, (e)), which has 13 normal and 3 reduction cells placed after the 3rd, 7th and 13th cells. ResNet-50's cells (Fig. 5, (d)) are similar to those of ResNet-18 (Fig. 5, (b)), but have 1×1 bottleneck layers.

Within and between cell connectivity. Within each cell and between them, there is a certain pattern to create connections in DARTS (Fig. 5, (b,d)): each cell receives features from the two previous cells, each summation node can only receive features from two nodes, the last concatenation node can receive features from an arbitrary number of nodes. But due to the presence of the Zero ('none') and Identity ('skip connection') operations, we can enable any connectivity. We represent operations as nodes[9] and drop redundant edges and nodes. For example, if the node performs the Zero operation, we remove the edges connected to that node. This can lead to a small fraction of disconnected graphs, which we remove from the training/testing sets. If the node performs the Identity operation, we remove the node, but keep corresponding edges. We also omit ReLU operations and other nonlinearities in a graph representation to avoid significantly enlarging graphs, since the position of nonlinearities w.r.t. other operations is generally consistent across architectures (e.g. all convolutions are preceded by ReLUs except the first layer). This leads to graphs visualized in Fig. 5, (c,e).

Operations. The initial choice of operations in DARTS is quite standard in NAS. In normal cells, each operation returns the tensor of the same shape as it receives. So any differentiable operation that can preserve the shape of the input tensor can be used, therefore extending the set of operations is relatively trivial. In reduction cells, spatial resolution is reduced by a factor of 2, so some operations can have stride 2 there. In both cells, there are summation and concatenation nodes that aggregate features from several operations into one tensor. Concatenation (across channels) is used as the final node in a cell. To preserve channel dimensions after concatenating many features, 1×1 convolutions are used as needed. For the Squeeze&Exicte (SE) and Visual Transformer (ViT) operations, we use open source implementations[10] with default configurations, e.g. 8 heads in the multihead self-attention of ViT.

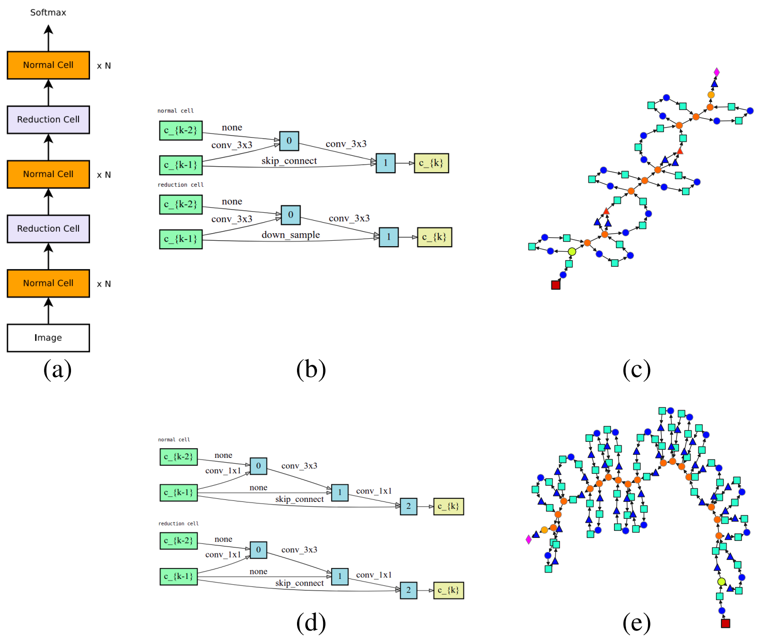


Figure 5: **(a)** Network's high-level structure introduced in [29] and employed by many following papers on network design, including DARTS [19] and ours, where N$\geq$ 1; **(b)** A residual block [8] in terms of DARTS normal and reduction cells, where green nodes denote outputs from the two previous cells, blue nodes denote summation, a yellow node denotes concatenation[†]; edges denote operations, 'none' indicates dropping the edge[‡]; the reduction cell has the same structure in ResNets, but decreases spatial resolution by 2 using a downsample operation and stride 2 in operations, at the same time, optionally increasing the channel dimensionality by 2. **(c)** The result of combining (a) and (b) for 8 cells using our code to build an analogous of the ResNet-18 architecture[⋆]. **(d)** A residual block of ResNet-50 with 1×1 bottleneck layers defined using DARTS and **(e)** the graph built using our code, where 3 reduction cells are placed as in the original ResNet-50 architecture.

[9] In DARTS, the operations are placed on the edges, except for inputs, summations and concatenation (Fig. 5).

[10] SE: `https://github.com/ai-med/squeeze_and_excitation/blob/master/squeeze_and_excitation/squeeze_and_excitation.py`, ViT: `https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py`

[†] Concatenation is redundant in ResNets and removed from our graphs due to only one input node in cells.

[‡] In ResNets [8], there is no skip connection between the input of a given cell and the output of the cell before the previous one.

[⋆] Note that ResNets of [8] commonly employed in practice have 3 reduction cells instead of 2 and have other minor differences (e.g. the order of convolution, BN and ReLU, down sampling convolution type, bottleneck layers, etc.). We still can predict parameters for them, but such architectures would be further away from the training distribution, so the predicted parameters might have significantly lower performance.

Table 7: Summary of differences between the DARTS design space and ours. [*]Means implementation-wise possibility of predicting parameters given a trained GHN, and does not mean our testing ID architectures, which follow the training design protocol. Overall, from the implementation point of view, our trained GHNs allow to predict parameters for arbitrary DAGs composed of our 15 primitives with the parameters of arbitrary shapes[¶]. We place ResNet-50 in a separate column even though it is one of the evaluation architectures of DEEPNETS-1M, because it has different properties as can be seen in the table.

PROPERTY	DARTS	DEEPNETS-1M	RESNET-50	TESTING GHN[*]
Unified style across image datasets	✗	✓	✗	✓
VGG style classification heads [25]	✗	✓	✗	✓
Visual Transformer stem [20]	✗	✓	✗	✓
Channel expansion ratio	2	1 or 2	2	arbitrary
Bottleneck layers (e.g. in ResNet-50 [8])	✗	✗	✓	✓
Reduction cells position (w.r.t. total depth)	1/3, 2/3	1/3, 2/3	3,7,17 cells	arbitrary
Networks w/o 1×1 preprocessing layers in cells	✗	✓	✓	✓
Networks w/o batch norm	✗	✓	✗	✓

A.2 DEEPNETS-1M Statistics

We show the statistics of the key properties of our DEEPNETS-1M in Fig. 6 and more examples of computational graphs for different subsets in Fig. 7.

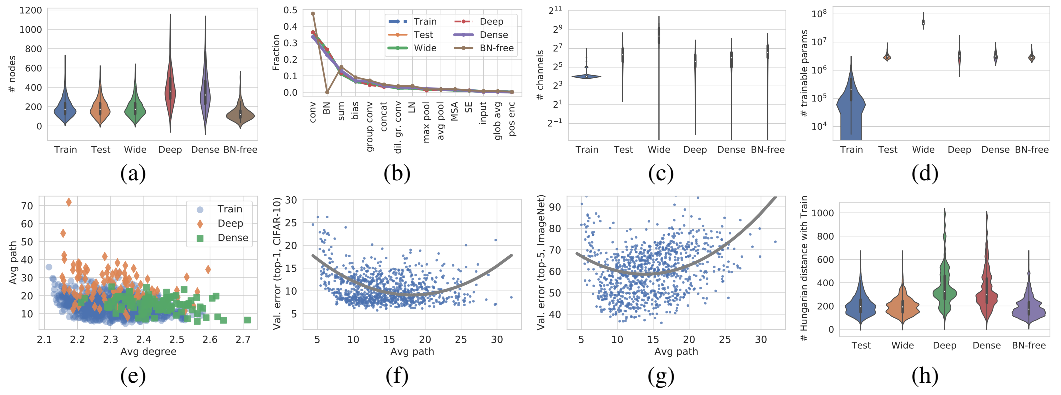


Figure 6: Visualized statistics of DEEPNETS-1M. (**a**) A violin plot of the number of nodes showing the distribution shift for the DEEP and DENSE subsets. (**b**) Node types (primitives) showing the distribution shift for the BN-FREE subset. (**c**) Number of initial channels in networks (for TRAIN, the number is for training GHNs on CIFAR-10). Here, the distribution shift is present for all test subsets (due to computational challenges of training GHNs on wide architectures), but the largest shift is for WIDE. (**d**) Total numbers of trainable parameters in case of CIFAR-10, where the distribution shifts are similar to those for the number of channels. (**e**) Average shortest path length versus average node degree (other subsets that are not shown follow the distribution of TRAIN), confirming that nodes of the DENSE subset have generally more dense connections (larger degrees), while in DEEP the networks are deeper (the shortest paths tend to be longer). (**f**) The validation error for 1000 VAL +TEST architectures (trained with SGD for 50 epochs) versus their average shortest path lengths, indicating the "sweet spot" of architectures with strong performance (same axes as in [46]). (**g**) Same as (f), but with the y axis being the top-5 validation error on ImageNet of the same architectures trained for 1 epoch according to our experiments. (**h**) The distribution of the distances between the architectures of a given test subset and the ones from a subset of TRAIN computed using the Hungarian algorithm [43], confirming that the evaluation architectures are different from the training ones.

[¶]While we predefine a wide range of possible shapes in our GHNs according to § B.1, in the rare case of using the shape that is not one of the predefined values, we use the closest values, which worked reasonably well in many cases.

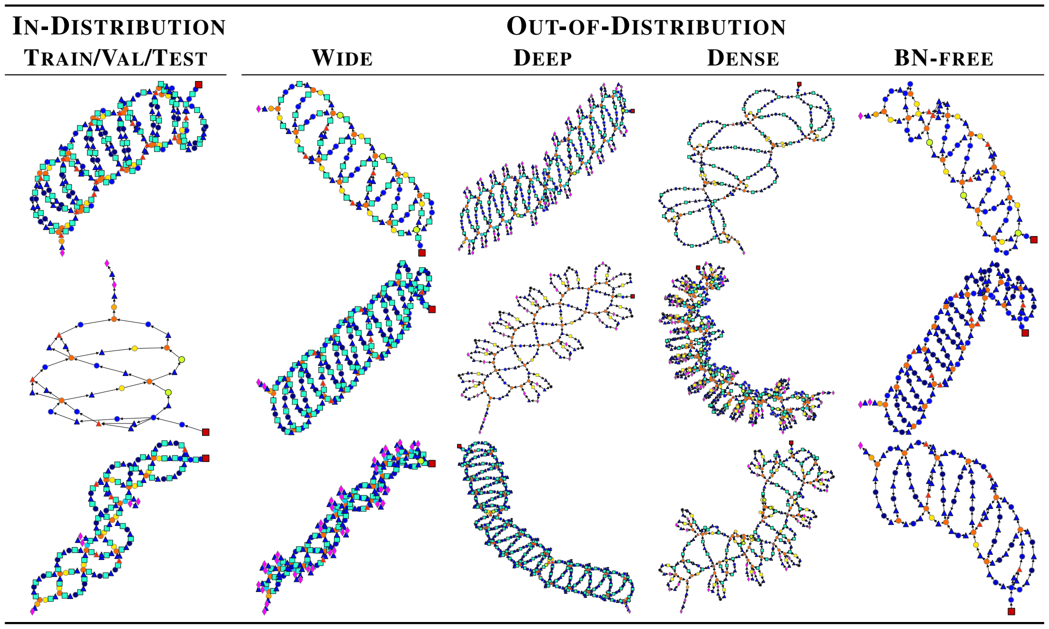


Figure 7: Examples of graphs in each subset of our DEEPNETS-1M visualized using NetworkX [44].

B GHN Details

B.1 Baseline GHN: GHN-1

GHNs were designed for NAS, which typically make strong assumptions about the choice of operations and their possible dimensions to make search and learning feasible. For example, non-separable 2D convolutions (e.g. with weights like $512\times512\times3\times3$ in ResNet-50) are not supported. Our parameter prediction task is more general than NAS, and tackling it using the vanilla GHNs of [24] is not feasible (mainly, in terms of GPU memory and training efficiency) as we show in § C.2.1 (Table 8). So we first make the following modifications to GHNs and denote this baseline as GHN-1.

1. **Compact decoder:** We support the prediction of full 4D weights of shape $<$*out channels* $\times$ *input channels* $\times$ *height* $\times$ *width*$>$ that is required for non-separable 2D convolutions. Using an MLP decoder of vanilla GHNs [24] would require it to have a prohibitive number of parameters (e.g. $\sim$4 billion parameters, see Table 8). To prevent that, we use an MLP decoder only to predict a small 3D tensor and then apply a 1×1 convolutional layer across the channels to increase their number followed by reshaping it to a 4D tensor.

2. **Diverse channel dimensions:** To enable prediction of parameters with channel dimensions larger than observed during training we implement a simple tiling strategy similar to [14]. In particular, instead of predicting a tensor of the maximum shape that has to be known prior training (as done in [24]), we predict a tensor with fewer channels, but tile across channel dimensions as needed. Combining this with #1 described above, our decoder first predicts a tensor of shape $128\times\mathcal{S}\times\mathcal{S}$, where $\mathcal{S}=11$ for CIFAR-10 and $\mathcal{S}=16$ for ImageNet. Then, the 1×1 convolutional decoder with 256 hidden and 4096 output units transforms this tensor to the tensor of shape $64\times64\times\mathcal{S}\times\mathcal{S}$. Modifications #1 and #2 can be viewed as strong regularizers that can hinder expressive power of a GHN and the parameters it predicts, but on the other side permit a more generic and efficient model with just around 1.6M-2M parameters.

3. **Fewer decoders:** One alternative strategy to reduce the number of parameters in the decoder of GHNs is to design multiple specialized decoders. However, this strategy does not scale well with adding new operations. Our modifications #1 and #2 allow us to have only three decoders: for convolutional and fully connected weights, for 1D weights and biases, such as affine transformations in normalization layers, and for the classification layer.

4. **Shape encoding:** The vanilla GHNs does not leverage the information about channel dimensionalities of parameters in operations. For example, the vanilla GHNs only differentiate between 3×3 and 5×5 convolutions, but not between $512 \times 512 \times 3 \times 3$ and $512 \times 256 \times 3 \times 3$. To alleviate that, we add two shape embedding layers: one for the spatial and one for the channel dimensions. For the spatial dimensions (height and width of convolutional weights) we predefine 11 possible values from 1 to S. For the channel dimensions (input and output numbers of channels) we predefine 392 possible values from 1 to 8192. We describe 3D, 2D and 1D tensors as special cases of 4D tensors using the value of 1 for missing dimensionalities (e.g. $10 \times 1 \times 1 \times 1$ for CIFAR-10 biases in the classification layer). The shape embedding layers transform the input shape into four (two for spatial and two for channel dimensions) 8-dimensional learnable vectors that are concatenated to obtain a 32-dimensional vector. This vector is summed with a 32-dimensional vector encoding one of the 15 operation types (primitives). In the rare case of feeding the shape that is not one of the predefined values, we look up for the closest value. This can work well in some cases, but can also hurt the quality of predicted parameters, so some continuous encoding as in [41] can be used in future work.

B.2 Our improved GHN: GHN-2

B.2.1 GHN-2 Architecture

Our improved GHN-2 is obtained by adding to GHN-1 the differentiable normalization of predicted parameters, virtual edges (and the associated 'mlp_ve' module, see the architecture below), meta-batching and layer normalization (and the associated 'ln' module). A high-level PyTorch-based overview of the GHN-2 model architecture used to predict the parameters for ImageNet is shown below (see our code for implementation details).

```
(ghn): GHN(  # our hypernetwork H_D with total 2,319,752 parameters (theta)
(gcn): GCNGated(  # Message Passing for Equations 3 and 4
  (mlp): MLP(
    (fc): Sequential(
      (0): Linear(in_features=32, out_features=16, bias=True)
      (1): ReLU()
      (2): Linear(in_features=16, out_features=32, bias=True)
      (3): ReLU()
    )
  )
  (mlp_ve): MLP(  # only in GHN-2 (see Eq. 4)
    (fc): Sequential(
      (0): Linear(in_features=32, out_features=16, bias=True)
      (1): ReLU()
      (2): Linear(in_features=16, out_features=32, bias=True)
      (3): ReLU()
    )
  )
  (gru): GRUCell(32, 32)   # all GHNs use d=32 for input, hidden and output node feature dimensionalities
)
(ln): LayerNorm((32,), eps=1e-05, elementwise_affine=True)  # only in GHN-2
(shape_embed_spatial): Embedding(11, 8)  # encodes the spatial shape of predicted params
(shape_embed_channel): Embedding(392, 8)  # encodes the channel shape of predicted params
(op_embed): Embedding(16, 32)  # 15 primitives plus one extra embedding for dummy nodes
(decoder): Decoder(
  (fc): Sequential(
    (0): Linear(in_features=32, out_features=32768, bias=True)
    (1): ReLU()
  )  # predicts a 128x16x16 tensor
  (conv): Sequential(
    (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(256, 4096, kernel_size=(1, 1), stride=(1, 1))
  )  # predicts a 64x64x16x16 tensor
```

```
      (class_layer_predictor): Sequential(
        (0): ReLU()
        (1): Conv2d(64, 1000, kernel_size=(1, 1), stride=(1, 1))
      )  # predicts 64x1000 classification weights given the 64x64x16x16 tensor
    )
    (norm_layers_predictor): Sequential(
      (0): Linear(in_features=32, out_features=64, bias=True)
      (1): ReLU()
      (2): Linear(in_features=64, out_features=128, bias=True)
    )  # predicts 1x64 weights and 1x64 biases given the 1x32 node embeddings
    (bias_predictor): Sequential(
      (0): ReLU()
      (1): Linear(in_features=64, out_features=1000, bias=True)
    )  # predicts 1x1000 classification biases given the 64x64x16x16 tensor
  )
```

B.2.2 Differentiable Normalization of Predicted Parameters

We analyze in more detail the effect of normalizing predicted parameters (Fig. 8).

Setup. As we discussed in § 4.1, Chang et al. [59] proposed a method to initialize a hypernetwork to stabilize the activations in the network for which the parameters are predicted. However, this technique requires knowing upfront the shapes of the predicted parameters, and therefore is not applicable out-of-the-box in our setting, where we predict the parameters of diverse architectures with arbitrary shapes of weights. So, instead we apply operation-dependent normalizations. We analyze the effect of this normalization by taking a GHN in the beginning and end of training on CIFAR-10 and predicting parameters of ResNet-50. To compute the variance of activations in the ResNet-50, we forward pass a batch of test images through the predicted parameters.

Observations. The activations obtained using GHN-1 explode after training, which aligns with the analysis of [59] (this is more obvious on the left of Fig. 8, where a linear scale is used on the y axis). For GHN-2, in the beginning of training the activations match the ones of ResNet-50 initialized randomly using Kaiming He's method [57], which validates the choice of our normalization equations in § 4.1. By the end of training, the activations of models for the random-based and the GHN-2-based cases decrease (perhaps, due to the weight decay), however, the ones of GHN-2 reduce less, indicating that the predicted parameters do not reach the state of those trained with SGD from scratch. In contrast, the activations corresponding to GHN-1 have small values in the beginning, but explode by the end of training. Matching the activations of the models trained with SGD can be useful to improve training of GHNs and, for example, to make fine-tuning of predicated parameters easier as we show in § 5.3, where the parameters predicted by GHN-1 are shown difficult to be fine-tuned with SGD.

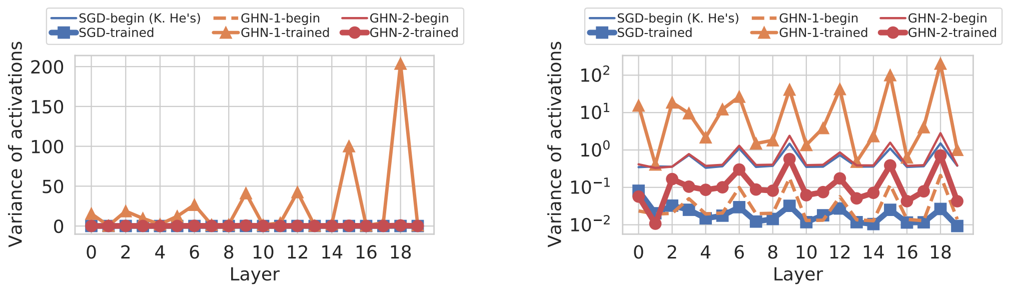


Figure 8: The effect of normalizing predicted parameters on the variance of activations in the first several layers of ResNet-50: a linear (**left**) and log (**right**) scale on the y axis.

B.2.3 Meta-batching

We analyze how meta-batching affects the training loss when training GHNs on CIFAR-10 (Fig. 9). The loss of the GHN-2 with $b_m = 8$ is less noisy and is lower throughout the training compared to using $b_m = 1$. In fact, the loss of $b_m = 1$ is unstable to the extent that oftentimes the training fails due to the numerical overflow, in which case we ignore the current architecture and resample a new one. For example, the standard deviation of gradient norms with $b_m = 8$ is significantly lower than with $b_m = 1$: 18 vs 145 with means 2.7 and 7.4 respectively. Training the model with $b_m = 1$ eight times longer (Fig. 9, right) boosts the performance of predicted parameters (Table 8), but still does not reach the level of $b_m = 8$; it also still suffers from the aforementioned numerical issues and does not leverage the parallelism of $b_m = 8$ (all architectures in a meta-batch can be processed in parallel). Further increasing the meta-batch size is an interesting avenue for future research.

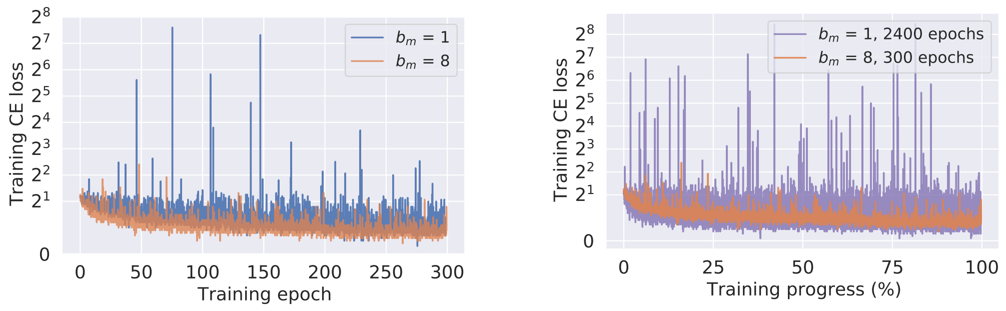


Figure 9: Effect of using more architectures per batch of images on the training loss (**left**) and comparison to training longer (**right**). In the figure on the right, we shrink the x axis of the $b_m = 1$ case, so that both plots can be compared w.r.t. the total number of epochs.

C Additional Experiments and Details

C.1 Experimental Details

Training GHNs. We train all GHNs for 300 epochs using Adam with an initial learning rate 0.001, batch size of 64 images for CIFAR-10 (256 for ImageNet), weight decay of 1e-5. The learning rate is decayed after 200 and 250 epochs. On ImageNet GHN-2's validation accuracy plateaued faster (in terms of training epochs), so we stopped training after 150 epochs decaying it after 100 and 140 epochs.

Evaluation of networks with BN using GHNs. While our GHNs predict all *trainable* parameters, batch norm networks have running statistics that *are not learned by gradient descent* [7], and so are not predicted by GHNs. To obtain these statistics, we evaluate the networks with BN by computing per batch statistics on the test set as in [24] using a batch size 64. Another alternative we have considered is updating the running statistics by forward passing several batches of the training or testing images through each evaluation network (no labels, gradients or parameter updates are required for this stage). For example on CIFAR-10 if we forward pass 200 batches of the training images (takes less than 10 seconds on a GPU), we obtain a higher accuracy of 71.7% on ID-TEST compared to 66.9% when the former strategy is used. On ImageNet, the difference between the two strategies is less noticeable. For both strategies, the results can slightly change depending on the order of images for which the statistics is estimated.

C.2 Additional Results

C.2.1 Additional Ablations

We present results on CIFAR-10 for different variants of GHNs (Table 8) in addition to those presented in Tables 3 and 5. Overall, different GHN variants show worse or comparable results on all evaluation architectures compared to our GHN-2, while in some cases having too many trainable parameters making training infeasible in terms of GPU memory or being less efficient to train (e.g. with $T = 5$ propagation steps). For ViT, GHN-2 is worse compared to other GHN variants, which requires further investigation.

Table 8: CIFAR-10 results of predicted parameters for the evaluation architectures of DEEPNETS-1M. Mean (±standard error of the mean) accuracies are reported. Different ablated GHN-2 models are evaluated. *GPU seconds per a batch of 64 images and b_m architectures. **In [24] the GHNs have fewer parameters due to a more constrained network design space (as discussed in B.1) and applying specialized decoders for different operations. The best result in each column is bolded, the best results with $b_m = 1$ (excluding training 8 × longer) are underlined.

MODEL	**Norm** $\hat{\mathbf{w}}_p$	**Virt. edges**	**M. batch** $b_m = 8$	**#GHN params**	**Train. speed***	**ID-TEST** avg	**ID-TEST** max	**OOD-TEST** WIDE	**OOD-TEST** DEEP	**OOD-TEST** DENSE	**OOD-TEST** BN-FREE	**OOD-TEST** RESNET/VIT
GHN-2	✓	✓	✓	1.6M	3.6	**66.9**±0.3	77.1	**64.0**±1.1	**60.5**±1.2	**65.8**±0.7	36.8±1.5	**58.6**/11.4
1000 archs	✓	✓	✓	1.6M	3.6	65.1±0.5	**78.4**	61.5±1.6	56.0±1.5	65.0±0.9	27.6±1.1	58.2/10.5
100 archs	✓	✓	✓	1.6M	3.6	47.1±0.8	77.1	38.8±1.9	28.3±1.6	41.9±1.5	11.0±0.2	38.7/10.3
No normalization	✗	✓	✓	1.6M	3.6	62.6±0.6	75.9	52.3±2.1	59.5±1.1	62.3±1.2	14.4±0.4	58.3/17.0
No virtual edges	✓	✗	✓	1.6M	3.6	61.5±0.4	73.2	58.2±1.0	55.0±0.9	61.5±0.6	**40.8**±0.8	41.9/12.1
No LayerNorm	✓	✓	✓	1.6M	3.6	64.5±0.4	75.9	62.4±1.1	59.0±1.1	64.6±0.6	39.6±1.2	55.1/8.9
No GatedGNN (MLP)	✓	✓	✓	1.6M	1.5	42.2±0.6	60.2	22.3±0.9	37.9±1.2	44.8±1.1	23.9±0.7	17.7/10.0
Train 8× longer	✓	✓	✗	1.6M	0.7	62.4±0.5	75.8	63.0±1.3	58.0±1.3	62.1±0.9	24.5±0.7	57.0/14.6
No Meta-batch ($b_m = 1$)	✓	✓	✗	1.6M	0.7	54.3±0.3	63.0	53.1±0.8	51.9±0.6	53.4±0.5	31.7±0.8	50.6/17.8
No Shape Encoding	✓	✓	✗	1.6M	0.7	53.1±0.4	61.7	52.4±0.9	51.4±0.7	53.5±0.6	24.7±0.8	31.5/14.0
No virtual edges (VE)	✓	✗	✗	1.6M	0.7	51.7±0.4	62.0	49.7±0.8	47.4±0.8	52.0±0.8	24.5±0.5	34.2/14.7
+Stacked GHN, no VE	✓	✗	✗	1.6M	0.7	52.2±0.4	63.6	51.3±0.9	46.9±0.9	52.3±0.7	20.2±0.6	44.5/15.4
+Stacked GHN	✓	✓	✗	1.6M	0.9	53.1±0.4	61.3	51.5±1.1	50.9±0.7	53.5±0.7	23.1±0.7	42.7/15.1
π = only fw (Eq. 3)	✓	✓	✗	1.6M	0.4	53.9±0.3	62.5	51.2±1.0	51.7±0.7	54.3±0.5	31.0±0.7	49.9/11.5
$T = 5$ (Eq. 3)	✓	✓	✗	1.6M	2.6	54.4±0.4	63.3	52.8±1.0	50.4±0.9	53.4±0.8	22.6±1.0	50.1/10.1
Fan-out	✓	✓	✗	1.6M	0.7	53.8±0.4	63.6	52.6±1.0	51.2±0.8	54.3±0.8	19.8±0.6	48.5/11.1
MLP decoder	✓	✓	✗	32M	0.7	53.1±0.4	64.0	52.9±1.0	52.5±0.7	54.0±0.8	22.1±0.5	44.1/16.3
No tiling	✓	✓	✗	135M		out of GPU memory						
No tiling, MLP decoder (as in a vanilla GHN [24])	✓	✓	✗	4.1B**		out of GPU memory						
GHN-1	✗	✗	✗	1.6M	0.6	51.4±0.4	59.9	43.1±1.7	48.3±0.8	51.8±0.9	13.7±0.3	19.2/**18.2**
GHN-1 + LayerNorm	✗	✗	✗	1.6M	0.6	50.1±0.5	58.9	43.8±1.4	47.5±0.8	50.8±1.0	11.4±0.2	49.2/16.3

C.2.2 Generalization Properties

On the OOD subsets, GHN results are lower than on ID-TEST as expected, so we inspect in more detail *how* performance changes with an increased distribution shift (Fig. 10). For example, training wider nets with SGD leads to similar or better performance, perhaps, due to increased capacity. However, GHNs degrade with larger width, since wide architectures are underrepresented during training for computational reasons (Table 1, Fig. 6). As for the depth and number of nodes, there is a certain range of values ("sweet spot") with higher performance. For architectures without batch norm (Fig. 10, the very right column), the results of GHN-2 are strong starting from a certain depth ($\geq$ 8-10 cells) matching the ones of training with SGD from scratch (for 50 epochs on CIFAR-10 and 1 epoch on ImageNet). This can be explained by the difficulty of training models without BN from scratch with SGD, while parameter prediction with GHN-2 is less affected by that. Generalization appears to be worse on ImageNet, perhaps due to its more challenging nature.

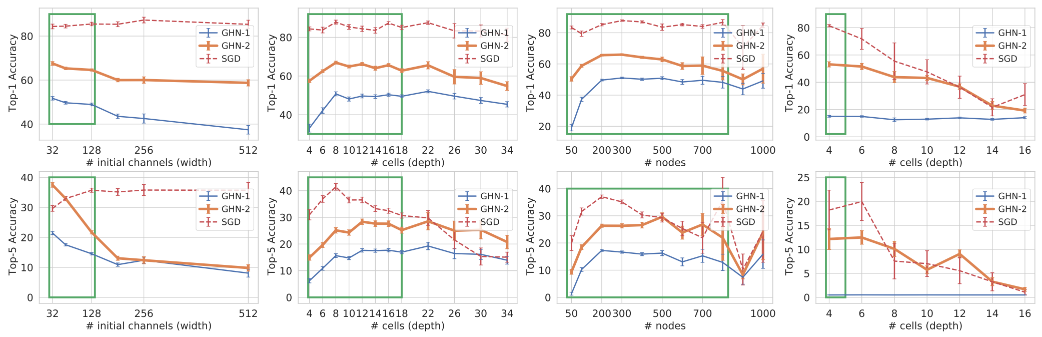
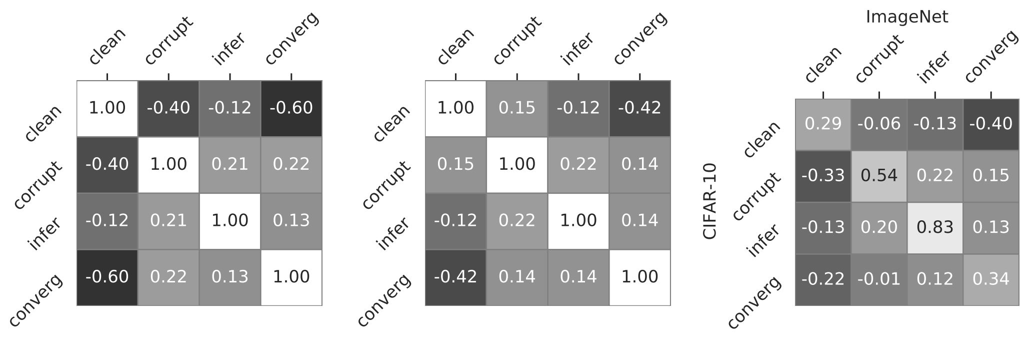


Figure 10: Generalization performance w.r.t. (from left to right): width, depth, number of nodes and depths for architectures without batch norm. A green rectangle denotes the training regime; bars are standard errors of the mean. Top row: CIFAR-10, bottom row: ImageNet.

C.2.3 Property Prediction

In § 5.2, we experiment with four properties of neural architectures that can be estimated given an architecture and image dataset:

1. "Clean" classification accuracy measured on the validation sets of CIFAR-10 and ImageNet.
2. Classification accuracy on the corrupted images, which is created by adding the Gaussian noise of medium intensity to the validation images following [53][11]: with zero mean and the standard deviation of 0.08 on CIFAR-10 and 0.18 on ImageNet.
3. The inference speed is measured as GPU seconds required to process the entire validation set.
4. For the convergence speed, we measure the number of SGD iterations to achieve a training top-1 accuracy of 95% on CIFAR-10 and top-5 accuracy of 10% on ImageNet.



CIFAR-10 **ImageNet** **CIFAR-10 vs ImageNet**

Figure 11: Cross correlation (Kendall's Tau) between ground truth values of properties.

Ground truth property values. We first obtain ground truth values for each property for each of the 500 ID-VAL and 500 ID-TEST architectures of DEEPNETS-1M trained from scratch with SGD for 50 epochs (on CIFAR-10) and 1 epoch (on ImageNet) as described in § 5. The ground truth values between these properties and between CIFAR-10 and ImageNet correlate poorly (Fig. 11). Interestingly, on CIFAR-10 the architectures that rank higher on the clean images generally rank lower on the corrupted set and vice verse (correlation is -0.40). On ImageNet the correlation between these two properties is positive, but low (0.15). Also, the transferability of architectures between CIFAR-10 and ImageNet is poor contrary to a common belief, e.g. with the correlation of 0.29 on the clean and -0.06 on the corrupted sets. However, this might be due to training on ImageNet only for 1 epoch. The networks that converge faster have generally worse performance on the clean set, but tend to perform better on the corrupted set. The networks that classify images faster (have higher inference speed), also tend to perform better on the corrupted set. Investigating the underlying reasons for these relationships as well as extending the set of properties would be interesting in future work.

Estimating properties by predicting parameters. A straightforward way to estimate this kind of properties using GHNs is by predicting the architecture parameters and forward passing images as was done in [24] for accuracy. However, this strategy has two issues: (a) the performance of parameters predicted by GHNs is strongly affected by the training distribution of architectures (Fig. 10); (b) estimating properties of thousands of networks for large datasets such as ImageNet can be time consuming. Regarding (a), for example the rank correlation on CIFAR-10 between the accuracy of the parameters predicted by GHN-2 and those trained with SGD is only 0.4 (down from 0.8 obtained with the regression model, Fig. 4).

Training regression models. To report the results in Fig. 4, we treat the 500 ID-VAL architectures of DEEPNETS-1M as the training ones in this experiment. We train a simple regression model for each property using graph embeddings (obtained using MLP, GHN-1 or GHN-2) and ground truth property values of these architectures. We use Support Vector Regression[12] with the RBF kernel and tune hyperparameters (C, gamma, epsilon) on these 500 architectures using 5-fold cross-validation.

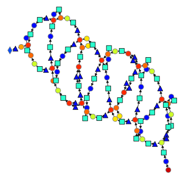
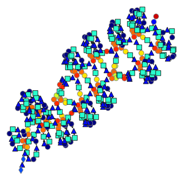
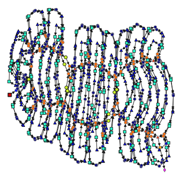
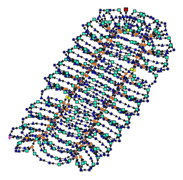
[11]https://github.com/hendrycks/robustness

[12]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

We then estimate the properties given a trained regression model on the 500 ID-Test architectures of DeepNets-1M and measure Kendall's Tau rank correlation with the ground truth test values. We repeat the experiment 5 times with different random seeds, i.e. different cross-validation folds. In Fig. 4, we show the average and standard deviation of correlation results across 5 runs. To train the graph convolutional network of Neural Predictor [80], we use the open source implementation[13] and train it on the 500 validation architectures from scratch for each property.

Downstream results. Next, we verify if higher correlations translate to downstream gains. We consider the clean accuracy on CIFAR-10 in this experiment as an example. We retrain the regression model on the graph embeddings of the combined ID-Val and ID-Test sets and generate 100k new architectures (similarly to how ID-Test are generated) to pick the most accurate one according to the trained regression model. We train the chosen architecture from scratch following [19, 24, 32], i.e. with SGD for 600 epochs using cutout augmentation (C), an auxiliary classifier (A) and the drop path (D) regularization. In addition, we train the chosen architecture for just 50 epochs using the same hyperparameters we used to train our ID-Val and ID-Test architectures as in § 5.1. We train the chosen architecture 5 times using 5 random seeds and report an average and standard deviation of the final classification accuracy on the test set of CIFAR-10 (Table 9). We perform this experiment for GHN-1 and GHN-2 in the same way. Among the methods we compare in Table 9, our GHN-2-based search finds the most performant network if training is done for 50 epochs (without and with C, A and D) and finds a competitive network if training is done for 600 epochs with C, A and D.

Table 9: CIFAR-10 best architectures and their performance on the test set. C — cutout augmentation, A — auxiliary classifier, D — drop path regularization. The best result in each row is bolded.

	GHN-1	**GHN-2**	**DARTS** [19]	**PDARTS** [32]
				
# params (M)	3.1	3.1	3.3	3.4
50 epochs	92.61±0.16	**93.94**±0.11	93.11±0.09	92.95±0.14
50 epochs + C,A,D [19]	91.80±0.14	**95.24**±0.14	94.50±0.08	94.22±0.06
600 epochs + C,A,D [19]	95.90±0.08	97.26±0.09	97.17±0.06	**97.48**±0.06

C.2.4 Comparing Neural Architectures

In addition to our experiments in § 5.2, we further evaluate representation power of GHNs by computing the distance between neural architectures in the graph embedding space.

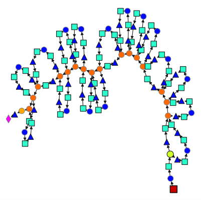
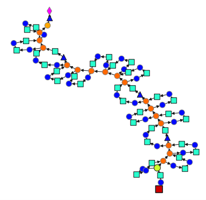
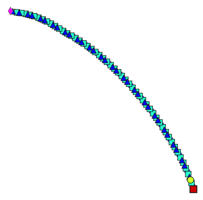
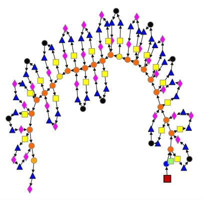
Experimental setup. We compute the pairwise distance between the computational graphs a_1 and a_2 as the ℓ_2 norm between their graph embedding $||\mathbf{h}_{a_1} - \mathbf{h}_{a_2}||$. We chose ResNet-50 as a reference network and compare its graph structure to the other three predefined architectures: ResNet-34, ResNet-50 without skip connections, and ViT. Among these, ViT is intuitively expected to have the largest ℓ_2 distance, because it is mainly composed of non-convolutional operations. ResNet-50-No-Skip should be closer (in the ℓ_2 sense) to ResNet-50 than ViT, because it is based on convolutions, but further away than ResNet-34, since it does not have skip connections.

Additional baseline. As a reference graph distance, we employ the Hungarian algorithm [43]. This algorithm computes the total assignment "cost" between two sets of nodes. It is used as an efficient approximation of the exact graph distance algorithms [106, 107], which are infeasible for graphs of the size we consider here.

Qualitative results. The ℓ_2 distances computed based on GHN-2 align well with our initial hypothesis of the relative distances between the architectures as well as with the Hungarian distances (Table 10). In contrast, the baselines inappropriately embed ResNet-50 closer to ResNet-50-No-Skip

[13]`https://github.com/ultmaster/neuralpredictor.pytorch`

Table 10: Comparing ResNet-50 to the three predefined architectures using the GHNs trained on CIFAR-10 in terms of the ℓ_2 distance between graph embeddings.

ResNet-50	ResNet-34	ResNet-50-No-Skip	ViT
			
Hungarian	118.0	134.0	207.5
MLP	0.4	0.2	1.0
GHN-1	0.6	0.5	1.7
GHN-2	1.0	1.3	3.1

than to ResNet-34. Thus, based on this simple evaluation, GHN-2 captures graph structures better than the baselines.

We further compare the architectures in the ID-Test set and visualize the most similar and dissimilar ones using our trained models. Based on the visualizations in Fig. 12, MLP as expected is not able to capture graph structures, since it relies only on node features (the most similar architectures shown on the left are quite different). The difference between GHN-1 and GHN-2 is hard to understand qualitatively, so we compare them numerically below.

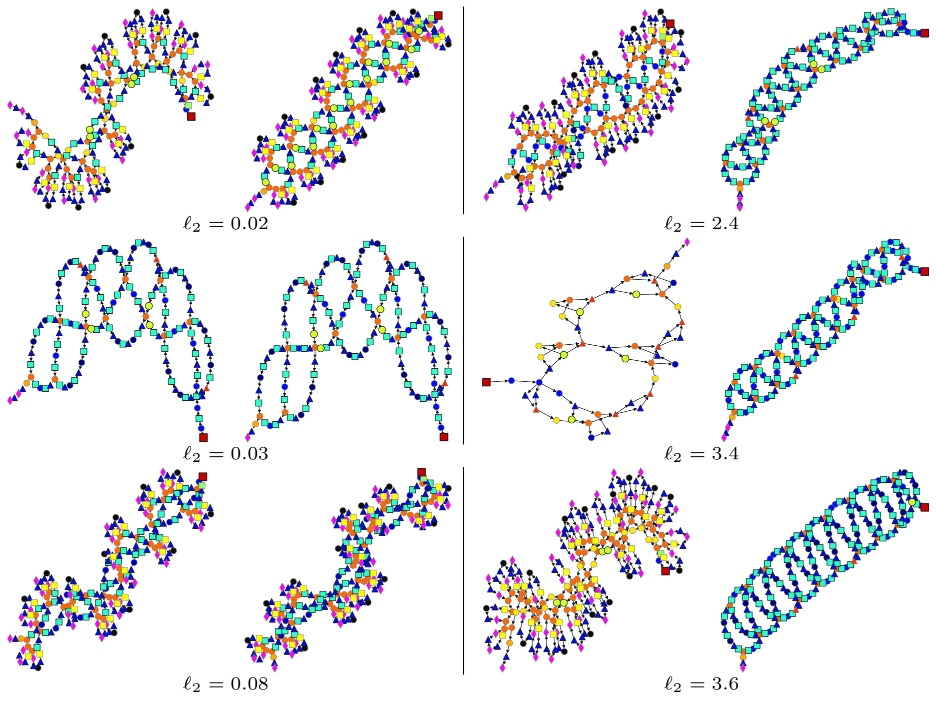


Figure 12: Most similar (left) and dissimilar (right) architectures (in terms of the ℓ_2 distance) in the ID-Test set based on the graph embeddings obtained by the MLP (top row), GHN-1 (middle row) and GHN-2 (bottom row) trained on CIFAR-10.

Quantitative results. To quantify the representation power of GHNs, we exploit the fact that, by design, the OOD architectures in our DeepNets-1M are more dissimilar from the ID architectures than the architectures within the ID distribution. So, a strong GHN should reflect that design principle and map the OOD architectures further from the ID ones in the embedding space. One way to measure the distance between two feature distributions, such as our graph embeddings, is the Fréchet distance (FD) [108]. We compute the FD between graph embeddings of 5000 training architectures and five test subsets (in the similar style as in [109–111]): ID-Test and four OOD subsets (Table 11).

GHN-2 maps the ID-TEST architectures close to the training ones, while the OOD ones are further away, reflecting the underlying design characteristics of these subsets. On the other hand, both baselines inappropriately map the DEEP and DENSE architectures as close or even closer to the training ones than ID-TEST, despite the differences in their graph properties (Table 1, Fig. 6). This indicates GHN-2's stronger representation compared to the GHN-1 and MLP baselines.

Table 11: FD between test and training graph embeddings on CIFAR-10. The average assignment cost between two sets of nodes using the Hungarian algorithm is shown for reference.

MODEL	ID-TEST	WIDE	DEEP	DENSE	BN-FREE
Hungarian	208.7	199.9	365.6	340.7	193.1
MLP	0.50	1.04	0.37	0.39	1.10
GHN-1	0.15	0.45	0.16	0.21	5.45
GHN-2	0.30	0.80	1.11	0.59	3.64

Alternative ways to compare graphs include running expensive graph edit distance (GED) algorithms, infeasible for graphs of our size, or designing task-specific graph kernels [78, 112, 77, 106]. As a more efficient (and less accurate) variant of GED, we employ the Hungarian algorithm. It finds the optimal assignment between two sets of nodes, so its disadvantage is that it ignores the edges. It still can capture the distribution shifts between graphs that can be detected based on the number of nodes and their features, such as for the DEEP and DENSE subsets (Table 11). It does not differentiate TRAIN and BN-FREE, perhaps, due to the fact that a small portion of architectures in TRAIN does not have BN. Finally, the inability of the Hungarian algorithm to capture the difference between TRAIN and WIDE can be explained by the fact that we ignore the shape of parameters when running this algorithm.

C.3 Analysis of Predicted Parameters

C.3.1 Diversity

We analyze how much parameter prediction is sensitive to the input network architecture. For that purpose, we analyze the parameters of 1,000 evaluation architectures (ID-VAL and ID-TEST) of DEEPNETS-1M on CIFAR-10. We compare the diversity of the parameters trained with SGD from scratch (He's initialization+SGD) to the diversity of the parameters predicted by GHNs. To analyze the parameters, we consider all the parameters associated with an operation, which is in general a 4D tensor (i.e. out channels $\times$ in channels $\times$ height $\times$ width). As tensor shapes vary drastically across architectures and layers, it is challenging to find a large set of tensors of the same shape. We found that the shape of $128\times1\times3\times3$ is one of the most frequent ones: appearing 760 times across the evaluation architectures. So for a given method of obtaining parameters (i.e. He's initialization+SGD or GHN), we obtain a set of 760 tensors with $128\times1\times3\times3 = 1152$ values in each tensor, i.e. a 760×1152 matrix. For each pair of rows in this matrix, we compute the absolute cosine distance, which results in a 760×760 matrix with values in range [0,1]. We report the mean value of this matrix in Table 12.

Results. The parameters predicted by GHN-2 are more diverse than the ones predicted by GHN-1: average distance is 0.17 compared to 0.07 respectively (Table 12). The parameters predicted by MLPs are extremely similar to each other, since the graph structure is not exploited by MLPs. The cosine distance is not exactly zero for MLPs, because two different primitives (group convolution and dilated group convolution) can have the same $128\times1\times3\times3$ shape. The parameters predicted by GHN-2 are more similar (low cosine distance) to each other compared to He's initialization+SGD. Low cosine distances in case of GHNs indicate that GHNs "store" good parameters to some degree However, our GHN-2 seems to rely on storing the parameters less than GHN-1 and MLP. Instead, GHN-2 relies more on the input graph to predict more diverse parameters depending on the layer and architecture. So, GHN-2 is more sensitive to the input graph. We believe this is achieved by our enhancements, in particular virtual edges, that allow GHN-2 to better propagate information across nodes of the graph.

C.3.2 Sparsity

We also analyze the sparsity of predicted parameters using the same 1,000 evaluation architectures. The sparsity can change drastically across the layers, so to fairly compare sparsities we consider

Table 12: Analysis of predicted parameters on CIFAR-10.

Method of obtaining parameters	Average distance between parameter tensors	Average sparsity
He's init. + SGD	0.98	33%
MLP	0.01	16%
GHN-1	0.07	20%
GHN-2	0.17	39%

the first layer only. We compute the sparsity of parameters $\mathbf{w}$ as the percentage of absolute values satisfying $|\mathbf{w}| < 0.05$. We report the average sparsity of all first-layer parameters in Table 12.

Results. The first-layer parameters predicted by GHN-2 are similar to the parameters trained by SGD in terms of sparsity: average sparsity is 39% compared to 33% respectively (Table 12). Higher sparsity of the parameters predicted by GHN-2 (39%) compared to the ones of GHN-1 (20%) may have been achieved due to the proposed parameter normalization method. The parameters predicted by GHN-2 are also more sparse than the ones obtained by SGD. Qualitatively, we found that GHN-2 predicts many values close to 0 in case of convolutional kernels 5×5 and larger, which is probably due to the bias towards more frequent 3×3 and 1×1 shapes during training. Mitigating this bias may improve GHN's performance.

C.4 Training Speed of GHNs

Training GHN-2 with meta-batch size takes 0.9 GPU hours per epoch on CIFAR-10 and around 7.4 GPU hours per epoch on ImageNet (Table 13). The training speed is mostly affected by the meta-batch size (b_m) and the sequential nature of the GatedGNN. Given that GHNs learn to model 1M architectures, the training is very efficient. The speed of training GHNs with can be further improved by better parallelization of the meta-batch. Note that GHNs need to be trained only once for a given image dataset. Afterwards, trained GHNs can be used to predict parameters for many arbitrary architectures in less than a second per architecture (see Table 4 in the main text).

Table 13: Times of training GHNs on NVIDIA V100-32GB using our code.

MODEL	# GPUs	CIFAR-10 64 images/batch		IMAGENET 256 images/batch	
		sec/batch	hours/epoch	sec/batch	hours/epoch
Training a single ResNet-50 with SGD	1	0.10	0.02	0.77	1.03
MLP with meta-batch size $b_m = 1$	1	0.32	0.06	0.67	0.90
GHN-2 with meta-batch size $b_m = 1$	1	1.16	0.23	1.54	2.06
GHN-2 with meta-batch size $b_m = 8$	1	7.17	1.40	out of GPU memory	
GHN-2 with meta-batch size $b_m = 8$	4	4.62	0.90	5.53	7.40

D Additional Related Work

Besides the works discussed in § 6, our work is also loosely related to other parameter prediction methods [113, 98, 114], analysis of graph structure of neural networks [46], knowledge distillation from multiple teachers [115], compression methods [116] and optimization-based initialization [117–119]. Denil et al. [113] train a model that can predict a fraction of network parameters given other parameters requiring to retrain the model for each new architecture. Bertinetto et al. [98] train a model that predicts parameters given a new few-shot task similarly to [18, 96], and the model is also tied to a particular architecture. The HyperGAN [114] allows to generate an ensemble of trained parameters in a computationally efficient way, but as the aforementioned works is constrained to a particular architecture. Finally, MetaInit [117], GradInit [118] and Sylvester-based initialization [119] can initialize arbitrary networks by carefully optimizing their initial parameters, but due to the optimization loop they are generally more computationally expensive compared to predicting parameters using GHNs. Overall, these prior works did not formulate the task nor proposed the methods of predicting performant parameters for diverse and large-scale architectures as ours.

Finally, the construction of our DEEPNETS-1M is related to the works on network design spaces. Generating arbitrary architectures using a graph generative model, e.g. [120, 121, 46], can be one way to create the training dataset $\mathcal{F}$. Instead, we leverage and extend an existing DARTS framework [19] specializing on neural architectures to generate $\mathcal{F}$. More recent works [122] or other domains [123] can be considered in future work.

E Limitations

Our work makes a significant step towards reducing the computational burden of iterative optimization methods. However, it is limited in several aspects.

Fixed dataset and objective. Compared to GHNs, one of the crucial advantages of iterative optimizers such as SGD is that they can be easily used to train on new datasets and new loss functions. Future work can thus focus on fine-tuning GHNs on new datasets and objectives or conditioning GHNs on data and hyperparameters akin to SGD.

Training speed. Training GHN-2 on larger datasets and with a larger meta-batch (b_m) becomes slower. For example, on ImageNet with $b_m = 8$ it takes about 7.4 hours per epoch using 4×NVIDIA V100-32GB using our PyTorch implementation. So, training of our GHN-2 on ImageNet for 150 epochs took about 50 days. The slow training speed is mainly due to limited parallelization of the meta-batch (i.e. using b_m GPUs should be faster) and the sequential nature of the GatedGNN.

Fine-tuning predicted parameters. While in § 5.3 we showed significant gains of using GHN-2 for initialization on two low-data tasks, we did not find such an initialization beneficial in the case of more data. In particular, we compare training curves of ResNet-50 on CIFAR-10 when starting from the predicted parameters versus from the random-based He's initialization [57]. For each initialization case, we tune hyperparameters such as an initial learning rate and weight decay. Despite ResNet-50 being an OOD architecture, GHN-2-based initialization helps the network to learn faster in the first few epochs compared to He's initialization (Fig. 13). However, after around 5 epochs, He's initialization starts to outperform GHN-2, diminishing its initial benefit. By fine-tuning the predicted parameters with Adam we could slightly improve GHN-2's results. However, the results are still worse than He's initialization in this experiment, which requires further investigation. Despite this limitation, GHN-2 significantly improves on GHN-1 in this experiment similarly to § 5.3.

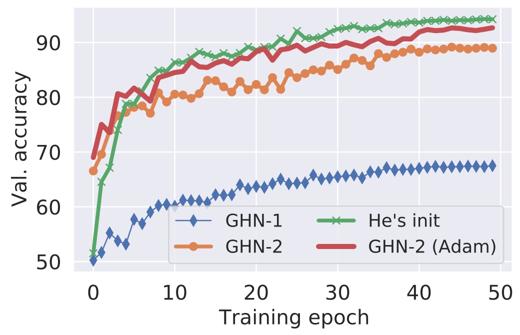


Figure 13: Training ResNet-50 on CIFAR-10 from random and GHN-based initializations.

F Societal Impact

One of the negative impacts of training networks using iterative optimization methods is the environmental footprint [11, 12]. Aiming at improving on the state-of-the-art, practitioners often spend tremendous computational resources, e.g. for manual/automatic hyperparameter and architecture search requiring rerunning the optimization. Our work can positively impact society by making a step toward predicting performant parameters in a resource-sustainable fashion. On the other hand, compared to random initialization strategies, in parameter prediction a single deterministic model could be used to initialize thousands or more downstream networks. All predicted parameters can inherit the same bias or expose any offensive or personally identifiable content present in a source dataset.