# Parameter Prediction for Unseen Deep Architectures

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Deep learning has been successful in automating the design of features and other critical components in machine learning pipelines. One critical element that remains largely hand-designed is the optimization algorithm used to iteratively optimize neural network parameters. Thus, a natural question is whether we can use deep learning to directly predict these parameters. To study this, we introduce a large-scale dataset of diverse 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 results on *unseen* and *diverse* networks. For example, it can predict all 24 million parameters of ResNet-50 achieving a 60% accuracy on CIFAR-10. We further scale our model to ImageNet, where networks with predicted parameters can achieve a top-5 accuracy close to 50%. As a useful byproduct, our model learns a strong representation of neural network architectures.

## 1 Introduction

Deep learning has exploded with training deep neural networks in a supervised fashion on large annotated datasets, such as ImageNet [1, 2]. The objective of such training can be defined as finding optimal parameters of an architecture $a$ (e.g. ResNet-50 [3]), parameterized by $w$ (e.g. convolutional weights), by minimizing a loss function $\mathcal{L}$ on a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ of inputs $\mathbf{x}_i$ and targets $y_i$:

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

where $f(\mathbf{x}_i; a, w)$ represents a forward pass. Equation 1 is usually minimized by iterative optimization algorithms – e.g. SGD [4] and Adam [5] – that converge to performant parameters $w_p$ of the architecture $a$. Despite the progress in improving the speed of these algorithms [6–8], they remain to be a bottleneck in large-scale machine learning pipelines, e.g. training ResNet-50 on ImageNet can take many GPU hours [9]. With the ever growing size of networks [10] and necessity of training them repeatedly (e.g. for hyperparameter or architecture search), the overall process of obtaining $w_p$ becomes computationally unsustainable [11–13].

In this paper, we aim to reduce the computational burden associated with iterative optimization by learning to predict performant parameters of arbitrary architectures. In particular, **we pose the task of designing a hypernetwork** [14] $H_{\mathcal{D}}$, operating in the domain $\mathcal{D}$ and parameterized by $\theta_p$, that takes as input a potentially *unseen* architecture $a$ and predicts a set of performant parameters $\hat{w_p}$ for this architecture: $\hat{w_p} = H_{\mathcal{D}}(a; \theta_p)$. A stretch goal for the hypernetwork is to predict $\hat{w_p}$ for which the test set performance of $f(\mathbf{x}; a, \hat{w_p})$ is similar to the one of $f(\mathbf{x}; a, w_p)$, where $w_p$ is obtained via standard iterative optimization algorithms. Approaching this task by collecting a training set of $\{a_i, w_{p,i}\}$ is prohibitive. Instead, we follow the bi-level optimization paradigm common in meta-learning [15–17],

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

but instead of iterating over $M$ tasks, we iterate over $M$ training architectures $\mathcal{F} = \{a_i\}_{i=1}^{M}$:

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

We argue that given recent advances in machine learning on graph-structured data [18–21], such as computational graphs representing $a$, the community is equipped with the tools to start tackling the task of predicting performant parameters for arbitrary architectures at scale. One particularly interesting work is Graph HyperNetworks (GHNs) [21] that optimizes Equation 2 in the context of neural architecture search (NAS). As such, GHNs focus on obtaining reliable subnetwork rankings rather than on predicting highly performant parameters for large-scale arbitrary architectures. As we show in our experiments, vanilla GHNs perform poorly in our parameter prediction task. We demonstrate that three modifications can drastically boost GHN performance: normalization of predicted parameters, enhanced long-range interactions, and meta-batching of architectures (Fig. 1). Our improved GHN model obtains an accuracy on *unseen* architectures of up to 77% (top-1) and 48% (top-5) on CIFAR-10 [22] and ImageNet [2], 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.

Overall, 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 model (§ 4) that performs surprisingly well on CIFAR-10 and ImageNet (§ 5.1); and (**d**) we show that our model learns a strong representation of neural network architectures (§ 5.2).

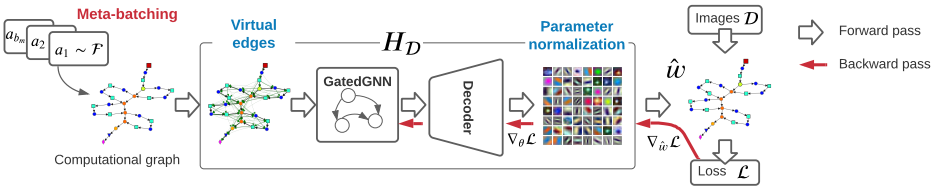



Figure 1: Overview of our model (§ 4) trained by backpropagation through the predicted parameters ($\hat{w}$) on a given image dataset and our DEEPNETS-1M dataset of architectures. Colored captions show our key improvements to vanilla GHNs (§ 2.2). The red one is used only during training GHNs, while the blue ones are used both at training and testing time. The computational graph of $a_1$ is visualized according to Table 1.

## 2 Background

To tackle the prediction of performant parameters (according to Equation 2), we need to introduce a dataset of architectures $\mathcal{F}$ as well as a hypernetwork $H_{\mathcal{D}}$. We start by providing a brief background about network design spaces, which we leverage in our DEEPNETS-1M creation in § 3. We follow by covering graph hypernetworks that we leverage when designing $H_{\mathcal{D}}$ in § 4.

### 2.1 Network Design Space of DARTS

DARTS [23] is a differentiable NAS framework. We are particularly interested in its network design space for image classification tasks 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: $3\times3$ and $5\times5$ separable convolutions, $3\times3$ and $5\times5$ dilated separable convolutions, $3\times3$ max pooling, $3\times3$ 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 fully connected layer. DARTS supports only one type of the classification head.

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, $1\times1$ convolutions are used as needed. All convolutional operations use the ReLU-Conv-Batch Norm (BN) [24] order. Overall, DARTS enables defining strong

architectures that combine many principles of manual [25, 3, 26, 27] and automatic [21, 28–33] design of neural architectures. While DARTS learns the optimal task-specific cells, the framework permits sampling randomly-structured cells which we leverage for the DEEPNETS-1M construction in § 3.

### 2.2 Graph HyperNetworks: GHN-1

In this subsection, we review the key components of GHNs [21] focusing on the representation of architectures $a$ and the design of the graph hypernetwork $H_{\mathcal{D}}$.

**Representation of** $a$. GHNs directly operate on the computational graph representation of a neural architecture $a$. Specifically, $a$ is a directed acyclyc 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. Such a representation is often employed by other NAS methods [34].

**Design of the graph hypernetwork** $H_{\mathcal{D}}$. In GHNs, $H_{\mathcal{D}}$ 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) [35]. In particular, their GatedGNN mimics the canonical order $\pi$ 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 is a multi-layer perceptron; and GRU is the update function of the Gated Recurrent Unit [36]. 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 outcoming 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 $\hat{w}_p^v$ (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 this model as GHN-1 (Fig. 1). Further subtleties of implementing this model in the context of our task are presented in the *Suppl. Mat.*, § B.1.

## 3 DEEPNETS-1M

The network design space of DARTS (described in § 2.1) is limited by the number of unique operations that compose cells, and the low variety of stems and classification heads. Thus, architectures such as VGG [25], ResNets [3], MobileNet [33] or more recent ones such as Visual Transformer (ViT) [37] and Normalization free networks [38, 39] are not realizable within this design space.

**Extending the network design space**. We extend the set of possible operations with non-separable 2D convolutions[1] (e.g. 3×3×512×512 as in ResNet-50), bias, Squeeze&Excite (SE) [40] (common in many efficient networks [41, 12]) and Transformer-based operations [42, 37]: multihead self-attention (MSA), positional encoding and layer norm (LN) [43]. Each node (operation) in our graphs has two attributes: *primitive type* (e.g. convolution) and *shape* (e.g. 3×3×512×512). 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, we allow the operations to not include batch norm (BN) [24] and permit networks without channel width expansion (e.g. as in [37]). Please see *Suppl. Mat.*, § A.1 and § A.2 for further details and statistics.

**Architecture generation process**. To generate architectures using the extended neural design space, 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 available operations. Finally, we filter out any disconnected graphs or networks with too many parameters (e.g. more than 10M and

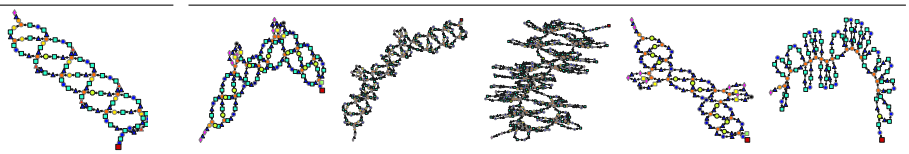
[1] NAS works, such as DARTS and GHN, avoid such convolutions, since the separable ones are more efficient.

100M parameters for training and testing respectively). Overall, we generate eight sets (Table 1). 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 [44] (in the same way as in § 5.2) to verify that.

**In-distribution (ID) architectures**. We generate the training set of $10^6$ architectures and the 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 our models we upper bound the number of parameters in training architectures to around 3M. In the evaluation sets, the number of channels is fixed. Therefore, in case of training our models, this results in some distribution shift from the training to the validation/test sets. However, the shift is not defined 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 wider and deeper nets in large-scale tasks [45, 46, 10], but which often suffer from more challenging training for fundamental [47, 48] or computational [49] reasons. 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 [50, 51] and the difficulty or unnecessity of using it in some cases [52–54, 38, 39]. We finally add the RESNET/VIT set with two predefined image classification architectures: ResNet-50 [3] and a smaller 12-layer version of the Visual Transformer (ViT) [37]. Our DEEPNETS-1M along with the network generator code will be made publicly available.

Table 1: Examples of computational graphs (visualized using NetworkX [55]) 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 [56, 57]. 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 $1\times1$ 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 ($\lvert V\rvert$) | 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%** | 100%/0% |
| #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 | 4.6±0.2 | 4.6±0.2 | 4.6±0.2 | 4.6±0.2 | **4.9**±0.2 | 4.8±0.2 | 4.5/4.6 |
| #avg path | 14.3±4.8 | 14.5±4.9 | 14.5±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 group 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 |

# 4 Improved Graph HyperNetworks: GHN-2

In this section, we introduce our three key improvements to the GHN-1 described in § 2.2 (Fig. 1). We analyze the effect of each improvement in *Suppl. Mat.*, § B.2 and empirically show their effectiveness in ablation studies in § 5 and further in *Suppl. Mat.*, § C.2.

## 4.1 Differentiable Normalization of Predicted Parameters

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 Kaiming's [58] or Xavier's [59] initialization to stabilize the variance of activations across layers of the network. Chang et al. [60] 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 this issue for our case of

predicting parameters of diverse architectures, we apply *operation-dependent normalizations*. We normalize convolutional and fully-connected weights by following the *fan-in* scheme of [58]: $\hat{w}_p^v \sqrt{\beta/(C_{in}\mathcal{H}\mathcal{W})}$, where $C_{in}, \mathcal{H}, \mathcal{W}$ are the number of input channels and spatial dimensions of weights $\hat{w}_p^v$, respectively; and $\beta$ is a nonlinearity specific constant following the analysis in [58], e.g. 2 for layers followed by ReLU. Alternatively, normalizing by output channels (*fan-out*) can be used. But as in [60, 58], in our case both strategies generally yield similar results. 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. In particular, we use $\text{sigmoid}(\hat{w}_p^v/T)$ for the weights and $\tanh(\hat{w}_p^v/T)$ for the biases. These are differentiable normalizations, so that they are applied at training (and testing) time. Our normalization stabilizes activations (see § B.2) improving training and the results.

### 4.2 Enhancing Long-range Message Propagation

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 [61]. Different approaches to alleviate the long-range propagation problem exist [62–64], including stacking GHNs in [21]. Instead we adopt simple graph-based heuristics in line with recent works [65, 66]. 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 $\text{MLP}_{\text{sp}}$ to transform the features of the nodes connected through these virtual edges, and redefine the message passing of Equation 3 as:

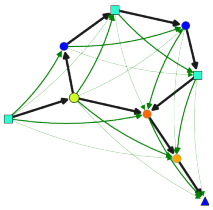


Figure 2: Example of adding virtual edges (in green) to better capture global context for a residual block.

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

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 the dimensionality of MLPs appropriately. Despite its simplicity, our approach compares favorably to stacking GHNs (§ C.2).

### 4.3 Meta-batching Architectures During Training

GHN-1 updates its parameters $\theta$ 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 [67]. Therefore, we define a meta-batch $b_m$ as the number of architectures sampled per batch of images. Note that both the parameter prediction as well as 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 $\theta$ of $H_{\mathcal{D}}$: $\nabla_\theta \mathcal{L} = 1/b_m \sum_{i=1}^{b_m} \nabla_\theta \mathcal{L}_i$, which improves stability of training and convergence speed (see *Suppl. Mat.*, § B.2).

## 5 Experiments

We focus the evaluation of GHN-2 on our parameter prediction task. In addition, we show beneficial side-effects of learning a stronger neural architecture representation using GHN-2. We provide further experimental and implementation details, as well as more results supporting our arguments in the *Suppl. Mat.*, § C. Our code will be made publicly available.

**Datasets**. We use the DEEPNETS-1M dataset of architectures (§ 3) as well as two image classification datasets $\mathcal{D}_1$ (CIFAR-10 [22]) and $\mathcal{D}_2$ (ImageNet [2]). CIFAR-10 consists of 50k training and 10k test images of size $32\times32\times3$ 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 $224\times224\times3$ following [23, 21]. 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 $\sim 50$ nodes and only trained on CIFAR-10, we reimplement[2] 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 in *Suppl. Mat.* for details). Otherwise, we follow the original GHNs [21].

[2]While source code for GHN [21] is not available, we appreciate the authors' help in implementing some steps.

**Comparisons to iterative optimizers**. We also compare our model to standard optimization methods: SGD and Adam [5]. We use off-the-shelf hyperparameters from GHN [21], given their ubiquitous use in the literature [23, 32, 68–70]. 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 [16, 17], 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 [71, 72], but are hard to reproduce without open source code.

**Additional experimental details**. All GHNs use $T = 1$ propagation (Equation 3), as we found the original $T = 5$ of [21] to be inefficient and not beneficial for our task. GHN-2 uses $s^{(\max)} = 50$ and $b_m = 8$ (introduced in § 4.2 and § 4.3 respectively) and additionally uses LN that slightly further improves results (see these ablations in *Suppl. Mat.*, § C.2). 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 Results on Parameter Prediction

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. We are interested in assessing 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 2) and ImageNet (Table 3). Despite never observing the test architectures, GHN-2 can predict all of their parameters in a single forward pass that perform surprisingly well on both image datasets. Our results are especially strong on CIFAR-10, where GHN-2 achieves up to 77.4% on some architectures, 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 48.1%. Surprisingly, even in the case of severe distribution shifts, i.e. for large architectures with bottleneck layers such as ResNet-50, and underrepresented ones such as ViT (it does not include BN and, except the first layer, convolutions – the two most frequent operations in the training set), our model still predicts parameters that perform better than random ones. On CIFAR-10, generalization of GHN-2 is particularly strong with 58.6% on ResNet-50.

On both image datasets, our GHN-2 is significantly better than 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 2). Exploiting the structure of computational graphs is a critical property of GHNs with the accuracy dropping from 66.7% to 43.2% 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 performs similarly to ∼2500 and ∼5000 iterations of SGD on CIFAR-10 and ImageNet respectively, while relying on a single forward pass to predict parameters. In contrast, GHN-1 performs similarly to ∼500 and ∼2000 (not shown in Table 3) iterations respectively.

To report speeds on ImageNet in Table 3, 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, SGD requires on average $\times 10^4$ more time on a GPU ($\times 10^5$ on a CPU) to obtain the parameters of perfor-

Table 2: CIFAR-10 results of predicted parameters for unseen ID and OOD architectures of DEEPNETS-1M. Mean (±standard error of the mean) accuracies are reported (random chance ≈10%).

| METHOD | #upd | ID-TEST | | OOD-TEST | | | | |
|---|---|---|---|---|---|---|---|---|
| | | avg | max | WIDE | DEEP | DENSE | BN-FREE | RESNET/VIT |
| MLP | 1 | 43.2±0.6 | 61.0 | 22.3±0.9 | 37.9±1.2 | 44.8±1.1 | 23.9±0.8 | 17.7/10.0 |
| GHN-1 | 1 | 50.8±0.5 | 59.4 | 43.1±1.7 | 48.3±0.8 | 51.8±0.9 | 13.7±0.3 | 19.2/**18.2** |
| GHN-2 | 1 | **66.7**±0.4 | **77.4** | **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) | $0.5\times10^3$ | 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) | $2.5\times10^3$ | 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) | $25\times10^3$ | 88.5±0.3 | 93.5 | 88.9±1.2 | 84.5±1.2 | 87.3±0.8 | 45.6±3.6 | 93.5/75.7 |
| Adam (50 epochs) | $25\times10^3$ | 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 3: ImageNet results of predicted parameters for unseen ID and OOD architectures of DEEPNETS-1M. Mean (±standard error of the mean) top-5 accuracies are reported (random chance ≈0.5%). *Estimated on ResNet-50 with batch size 128.

| METHOD | #upd | GPU sec. avg | CPU sec. avg | ID-TEST avg | ID-TEST max | OOD-TEST WIDE | OOD-TEST DEEP | OOD-TEST DENSE | OOD-TEST BN-FREE | OOD-TEST RESNET/VIT |
|---|---|---|---|---|---|---|---|---|---|---|
| GHN-1 | 1 | 0.3 | 0.5 | 16.8±0.4 | 32.3 | 15.7±0.9 | 15.8±0.8 | 15.1±0.8 | 0.5±0.0 | **6.8**/0.9 |
| GHN-2 | 1 | 0.3 | 0.7 | **27.2**±0.6 | **48.1** | **19.3**±1.4 | **24.7**±1.4 | **26.4**±1.2 | **7.2**±0.6 | 5.0/**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 | $2\times10^3$ | $3\times10^4$ | 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 | $4\times10^3$ | $6\times10^4$ | 37.0±0.4 | 62.0 | 38.9±1.6 | 22.1±1.5 | 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 |

Table 4: 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 | **66.7**±0.4 | **56.8**±0.8 | **1.9** |
| 1000 training architectures | 65.1±0.5 | 52.5±1.0 | 2.6 |
| No normalization (§ 4.1) | 62.6±0.6 | 47.1±1.2 | 3.9 |
| No virtual edges (§ 4.2) | 61.5±0.4 | 53.9±0.6 | 4.1 |
| No meta-batch ($b_m = 1$, § 4.3) | 54.3±0.3 | 47.5±0.6 | 5.5 |
| $b_m = 1$, train 8× longer | 62.4±0.5 | 51.9±1.0 | 3.7 |
| No GatedGNN (MLP) | 43.2±0.6 | 32.2±0.7 | 7.4 |
| GHN-1 | 50.8±0.5 | 39.2±0.9 | 6.8 |

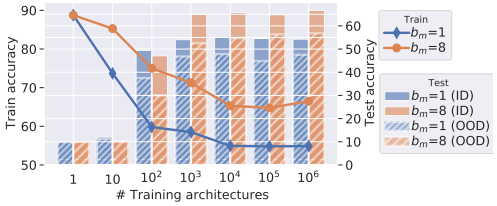



Figure 3: GHN-2 with meta batch $b_m = 8$ versus $b_m = 1$ for different numbers of training architectures on CIFAR-10.

mance similar to GHN-2. As a concrete example, AlexNet [1] requires around 50 GPU hours (on our setup) to achieve 81.8% while on some architectures we achieve 48.0% in just 0.3 GPU seconds.

Ablations (Table 4) show that all the 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 *Suppl. Mat.*, § C.2). Using the meta-batch ($b_m = 8$) is also essential and helps stabilizing training and speeds up convergence (see *Suppl. Mat.*, § B.2). To confirm that this performance gap is not primarily due to the observation of more architectures, we trained the model with $b_m = 1$ eight times longer and in Table 4 show that it is still inferior to $b_m = 8$. 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 the meta-batch starts to overfit to the training architectures. Surprisingly, using 1k training architectures already gives strong results, but leveraging all 1M architectures is still beneficial, to a larger extent for OOD generalization. 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 Representation of Neural Architectures

To solve our parameter prediction task better, the GHNs should hypothetically capture the structure of graphs better, which is a challenging problem in general [74–77]. We begin by validating this hypothesis quantitatively. In particular, we attempt to predict the properties of architectures given their graph embeddings obtained by averaging node features[3]. We consider four such properties: (1) accuracy on the clean test set, (2) accuracy on a corrupted test set (obtained by adding the Gaussian noise to images following [51]), (3) inference speed (latency or GPU seconds per a batch of images), and (4) speed of convergence (the number of SGD iterations to achieve a certain training accuracy). These properties correlate poorly with each other and between CIFAR-10 and ImageNet, so they

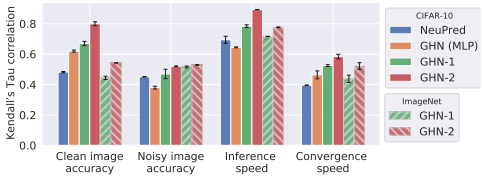
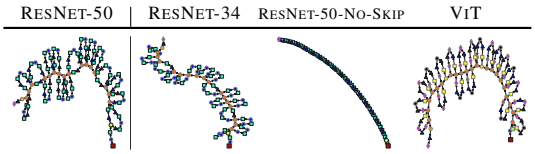



Figure 4: Property prediction of neural networks in terms of correlation (higher is better). Error bars denote the standard deviation across 5 runs.

[3] 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.

require the model to capture different regularities of graphs. We refer to *Suppl. Mat.*, § C.2 for a detailed description of these properties and relationships between them. While specialized methods to estimate some of these properties exist, often as a NAS task [34, 78, 79, 30, 74], 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 of 500 validation architectures (see *Suppl. Mat.*, § C.2 for details). We then measure Kendall's Tau rank correlation on 500 testing architectures between the predicted and ground truth property values similar to the Neural Predictor (NeuPred) of [34]. GHN-2 consistently outperforms the GHN-1 and MLP baselines as well as NeuPred developed for accuracy prediction (Fig. 4). We train a separate such NeuPred for each property from scratch following their hyperparameters. In *Suppl. Mat.*, § C.2 we also provide results of 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 [23, 21], we obtained a 97.26%($\pm$0.09) accuracy, which is competitive with leading NAS approaches, e.g. [23, 21, 32, 68–70]. In contrast, the network picked by the regression model trained on the GHN-1 embeddings achieves 95.90%($\pm$0.08).

**Qualitative results.** We analyze the representation power of GHNs qualitatively by computing the distance between graphs $a_1$ and $a_2$ as the $l_2$ norm between their graph embedding $||\mathbf{h}_{a,1} - \mathbf{h}_{a,2}||$. We compare ResNet-50 to the other three predefined architectures: ResNet-34, ResNet-50 without skip connections, and ViT. Among these, ViT is expected to have the largest distance because it is mainly composed of non-convolutional operations. ResNet-50-No-Skip should be closer to ResNet-50 than ViT, because it is based on convolutions, but further away than ResNet-34, since it does not have skip connections. We confirm our intuition by computing the graph distance based on the Hungarian algorithm [44], often employed as a more efficient approximation of exact graph distance algorithms [80, 81]. This algorithm returns the total assignment "cost" between two sets of nodes. Our GHN-2 appears to capture the differences between graph structures better than the baselines (Fig. 5). This further confirms that better solving our task translates to more powerful graph embeddings. We numerically analyze graph similarities more in *Suppl. Mat.*, § C.2.3.




| | 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 |

Figure 5: Comparing ResNet-50 to the three predefined architectures based on the GHNs trained on CIFAR-10 in terms of the $l_2$ distance between graph embeddings.

## 6 Related Work

Our proposed parameter prediction task and the models approaching it are related to a wide range of machine learning frameworks. We briefly discuss only the most representative of them.

Meta-learning is a general framework [15, 82] that includes meta-optimizers and meta-models, among others. **Meta-optimizers** [16, 17, 72, 83] define a problem similar to our task, but where $H_{\mathcal{D}}$ is an RNN-based model predicting the gradients $\nabla w$, mimicking the behavior of iterative optimizers. Therefore, the latter 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 GPU) for many iterations following Equation 1. **Meta-models** include methods based on MAML [84], ProtoNets [85] and auxiliary nets predicting task-specific parameters [86–88]. 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 [89] combines MAML with DARTS [23] 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 [90] makes 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 [91] 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 (image dataset), but aims at generalization *across architectures* $a$ with the ability to predict parameters in a single forward pass.

**One-shot NAS** aims to learn a single "supernet" [92, 12, 93] 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) [92] eliminate the need to train subnets. However, the fundamental limitation of one-shot NAS is poor scaling with the number of possible computational operations [21]. 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 [94] proposed to predict parameters using hypernetworks [14] by encoding architectures as a 3D tensor. Graph HyperNetworks (GHNs) [21] further generalize this approach to "arbitrary" computational graphs (DAGs), which allowed them to improve NAS results. Conventionally, GHNs are trained in a quite constrained architecture space [95]. We relax the architecture space adopting GHNs for a more general problem.

**Design spaces** Generating arbitrary architectures using a graph generative model, e.g. [96, 97, 57], can be one way to create the training dataset $\mathcal{F}$. We leverage an existing DARTS framework [23] to generate strong and diverse nets. More recent works [98] or other domains [99] can be considered.

Our work is also loosely related to other parameter prediction methods [100–102], analysis of graph structure of neural networks [57], knowledge distillation from multiple teachers [103] and compression methods [104]. However, these did not provide methods to predict parameters for such diverse and large-scale architectures as ours.

## 7 Limitations

Our work makes a significant step towards reducing the computational burden of iterative optimization methods. However, it is limited in several aspects. **Fine-tuning predicted parameters:** We verified if we can benefit from fine-tuning the predicted parameters with SGD to further improve their accuracy. We use CIFAR-10 and train ResNet-50 for 50 epochs as for other evaluation architectures and tune hyperparameters such as an initial learning rate, weight decay and SGD momentum. Despite ResNet-50 being an OOD architecture, starting from our predicted parameters helps it converge faster in the first few epochs compared to the random-based initialization of He et al. [58], which is standard for ResNets (Fig. 6). However, after around 5 epochs, He's initialization starts to outperform GHN-2. By fine-tuning the predicted parameters with Adam we could slightly improve the results, but we are still far from seeing a net benefit of using GHN-2 for initialization. This requires further investigation. Despite this limitation, GHN-2 significantly improves on GHN-1 in this experiment. **Fixed dataset and objective:** Compared to GHNs, one of the crucial advantages of iterative optimizers is that they can be 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 ImageNet using our PyTorch implementation is slow with around 7.5 hours/epoch using 4×NVIDIA V100-32GB. The slow training speed is mainly due to using the meta-batch and the sequential nature of the GatedGNN.

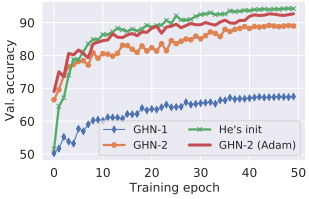



Figure 6: Training ResNet-50 on CIFAR-10 from random and GHN-based initializations.

## 8 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.

## 9 Conclusion

We propose a novel framework and benchmark to learn and evaluate neural parameter prediction models. Our model is able to predict parameters for very diverse and large-scale architectures in a single forward pass in a fraction of a second. These parameters yield surprisingly high accuracy given the extremely challenging nature of the task. However, the accuracy is still far from networks trained with handcrafted optimization methods. Bridging the gap is a promising future direction. As an interesting side-effect, our models learn a strong representation of neural architectures as evidenced by our property prediction evaluation. This motivates further research towards solving our task.

# References


[1] 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.

[2] 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.

[3] 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.

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

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

[6] 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.

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

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

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

[16] 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.

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

[18] 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`.

[19] 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`.

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

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

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

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

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

[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] 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.

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

[36] 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.

[37] 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.

[38] 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.

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

[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] 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.

[42] 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.

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

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

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

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

[47] 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.

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

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

[50] 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.

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

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

[53] 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.

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

[55] 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.

[56] 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.

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

[58] 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.

[59] 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.

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

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

[62] 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.

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

[64] 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.

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

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

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[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] 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.

[75] 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.

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

[77] 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.

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

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

[80] 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.

[81] 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.

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

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

[84] 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.

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

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

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

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

[95] 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.

[96] 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.

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

[98] 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.

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

[100] 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.

[101] 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.

[102] 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.

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

[104] 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.

[105] 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.

[106] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[107] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International Conference on Machine Learning*, pages 3835–3845. PMLR, 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] 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.

# Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See experiments in § 5.
   (b) Did you describe the limitations of your work? [Yes] See § 7.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See § 8.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code and the datasets will be released upon publication.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See § 5 and *Suppl. Mat.*
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See captions to Tables 2, 3, and Fig. 4.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See § 7.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] DARTS [23], CIFAR-10 [22], ImageNet [2]
   (b) Did you mention the license of the assets? [Yes] DARTS license is Apache License 2.0
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] The code and the datasets will be released upon publication.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We use CIFAR-10 and ImageNet, which are widely used benchmarks.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] See § 8.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]