RAPID NEURAL ARCHITECTURE SEARCH BY LEARNING TO GENERATE GRAPHS FROM DATASETS

Hayeon Lee[1]*, **Eunyoung Hyung**[1]*, **Sung Ju Hwang**[1,2]
KAIST[1], AITRICS[2], South Korea
{hayeon926, eunyoung0301, sjhwang82}@kaist.ac.kr

ABSTRACT

Despite the success of recent Neural Architecture Search (NAS) methods on various tasks which have shown to output networks that largely outperform human-designed networks, conventional NAS methods have mostly tackled the optimization of searching for the network architecture for a single task (dataset), which does not generalize well across multiple tasks (datasets). Moreover, since such task-specific methods search for a neural architecture from scratch for every given task, they incur a large computational cost, which is problematic when the time and monetary budget are limited. In this paper, we propose an efficient NAS framework that is trained once on a database consisting of datasets and pretrained networks and can rapidly search for a neural architecture for a novel dataset. The proposed MetaD2A (Meta Dataset-to-Architecture) model can stochastically generate graphs (architectures) from a given set (dataset) via a cross-modal latent space learned with amortized meta-learning. Moreover, we also propose a meta-performance predictor to estimate and select the best architecture without direct training on target datasets. The experimental results demonstrate that our model meta-learned on subsets of ImageNet-1K and architectures from NAS-Bench 201 search space successfully generalizes to multiple unseen datasets including CIFAR-10 and CIFAR-100, with an average search time of 33 GPU seconds. Even under MobileNetV3 search space, MetaD2A is 5.5K times faster than NSGANetV2, a transferable NAS method, with comparable performance. We believe that the MetaD2A proposes a new research direction for rapid NAS as well as ways to utilize the knowledge from rich databases of datasets and architectures accumulated over the past years. Code is available at https://github.com/HayeonLee/MetaD2A.

1 INTRODUCTION

The rapid progress in the design of neural architectures has largely contributed to the success of deep learning on many applications (Krizhevsky et al., 2012; Cho et al., 2014; He et al., 2016; Szegedy et al.; Vaswani et al., 2017; Zhang et al., 2018). However, due to the vast search space, designing a novel neural architecture requires a time-consuming trial-and-error search by human experts. To tackle such inefficiency in the manual architecture design process, researchers have proposed various *Neural Architecture Search (NAS)* methods that automatically search for optimal architectures, achieving models with impressive performances on various tasks that outperform human-designed counterparts (Baker et al., 2017; Zoph & Le, 2017; Kandasamy et al., 2018; Liu et al., 2018; Luo et al., 2018; Pham et al., 2018; Liu et al., 2019; Xu et al., 2020; Chen et al., 2021).

Recently, large benchmarks for NAS (NAS-101, NAS-201) (Ying et al., 2019; Dong & Yang, 2020) have been introduced, which provide databases of architectures and their performances on benchmark datasets. Yet, most conventional NAS methods cannot benefit from the availability of such databases, due to their task-specific nature which requires repeatedly training the model from scratch for each new dataset (See Figure 1 Left). Thus, searching for an architecture for a new task (dataset) may require a large number of computations, which may be problematic when the time and monetary budget are limited. How can we then exploit the vast knowledge of neural architectures that have

*These authors contributed equally to this work.

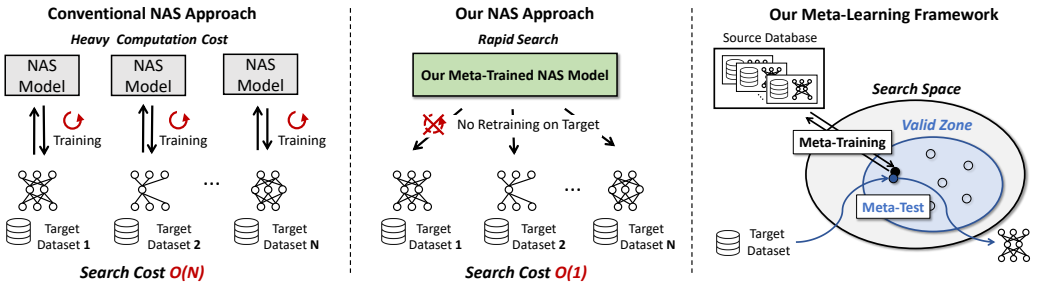

Figure 1: **Left:** Most conventional NAS approaches need to repeatedly train NAS model on each given target dataset, which results in enormous total search time on multiple datasets. **Middle:** We propose a novel NAS framework that generalizes to any new target dataset to generate specialized neural architecture without additional NAS model training after only meta-training on the source database. Thus, our approach cut down the search cost for training NAS model on multiple datasets from O(N) to O(1). **Right:** For unseen target dataset, we utilize amortized meta-knowledge represented as set-dependent architecture generative representations.

been already trained on a large number of datasets, to better generalize over an unseen task? In this paper, we introduce amortized meta-learning for NAS, where the goal is to learn a NAS model that generalizes well over the task distribution, rather than a single task, to utilize the accumulated meta-knowledge to new target tasks. Specifically, we propose an efficient NAS framework that is trained once from a database containing datasets and their corresponding neural architectures and then generalizes to multiple datasets for searching neural architectures, by learning to generate a neural architecture from a given dataset. The proposed MetaD2A (Meta Dataset-to-Architecture) framework consists of a set encoder and a graph decoder, which are used to learn a cross-modal latent space for datasets and neural architectures via amortized inference. For a new dataset, MetaD2A stochastically generates neural architecture candidates from set-dependent latent representations, which are encoded from a new dataset, and selects the final neural architecture based on their predicted accuracies by a performance predictor, which is also trained with amortized meta-learning. The proposed meta-learning framework reduces the search cost from O(N) to O(1) for multiple datasets due to no training on target datasets. After one-time building cost, our model only takes just a few GPU seconds to search for neural architecture on an unseen dataset (See Figure 1).

We meta-learn the proposed MetaD2A on subsets of ImageNet-1K and neural architectures from the NAS-Bench201 search space. Then we validate it to search for neural architectures on multiple unseen datasets such as MNIST, SVHN, CIFAR-10, CIFAR-100, Aircraft, and Oxford-IIIT Pets. In this experiment, our meta-learned model obtains a neural architecture within 33 GPU seconds on average without direct training on a target dataset and largely outperforms all baseline NAS models. Further, we compare our model with representative transferable NAS method (Lu et al., 2020) on MobileNetV3 search space. We meta-learn our model on subsets of ImageNet-1K and neural architectures from the MobileNetV3 search space. The meta-learned our model successfully generalizes, achieving extremely fast search with competitive performance on four unseen datasets such as CIFAR-10, CIFAR-100, Aircraft, and Oxford-IIIT Pets.

To summarize, our contribution in this work is threefold:

- We propose a novel NAS framework, MetaD2A, which rapidly searches for a neural architecture on a new dataset, by sampling architectures from latent embeddings of the given dataset then selecting the best one based on their predicted performances.

- To this end, we propose to learn a cross-modal latent space of datasets and architectures, by performing amortized meta-learning, using a set encoder and a graph decoder on subsets of ImageNet-1K.

- The meta-learned our model successfully searches for neural architectures on multiple unseen datasets and achieves state-of-the-art performance on them in NAS-Bench201 search space, especially searching for architectures within 33 GPU seconds on average.

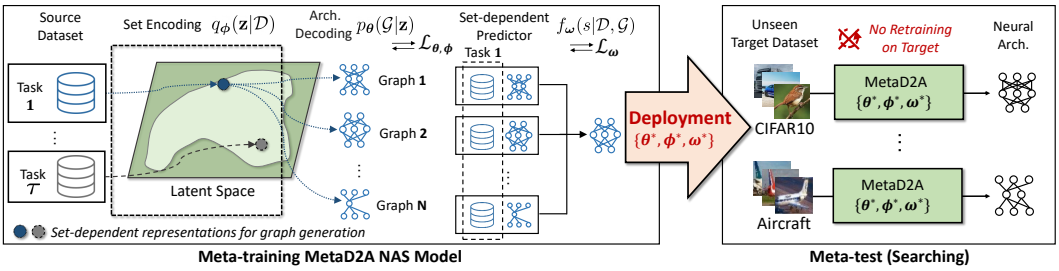

Figure 2: **Overview of MetaD2A** The proposed generator with $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ meta-learns *the set-dependent graph representations* on the meta-training tasks, where each task contains a subset of ImageNet-1K and high-quality architecture for the subset. The proposed predictor with $\boldsymbol{\omega}$ meta-learns to predict performance, considering the dataset as well as the graph. In the meta-test (searching) phase, the meta-learned MetaD2A generalizes to output set-specialized neural architecture for new target datasets without additional NAS model training.

2 Related Work

Neural Architecture Search (NAS) NAS is an automated architecture search process which aims to overcome the suboptimality of manual architecture designs when exploring the extensive search space. NAS methods can be roughly categorized into reinforcement learning-based methods (Zoph & Le, 2017; Zoph et al., 2018; Pham et al., 2018), evolutionary algorithm-based methods (Real et al., 2019; Lu et al., 2020), and gradient-based methods (Liu et al., 2019; Cai et al., 2019; Luo et al., 2018; Dong & Yang, 2019b; Chen et al., 2021; Xu et al., 2020; Fang et al., 2020). Among existing approaches, perhaps the most relevant approach to ours is NAO (Luo et al., 2018), which maps DAGs onto a continuous latent embedding space. However, while NAO performs graph reconstruction for a single task, ours generates data-dependent Directed Acyclic Graphs (DAGs) across multiple tasks. Another important open problem in NAS is reducing the tremendous computational cost resulting from the large search space (Cai et al., 2019; Liu et al., 2018; Pham et al., 2018; Liu et al., 2019; Chen et al., 2021). GDAS (Dong & Yang, 2019b) tackles this by optimizing sampled sub-graphs of DAG. PC-DARTS (Xu et al., 2020) reduces GPU overhead and search time by partially selecting channel connections. However, due to the task-specific nature of those methods, they should be retrained from the scratch for each new unseen task repeatedly and each will take a few GPU hours. The accuracy-predictor-based transferable NAS called NSGANetV2 (Lu et al., 2020) alleviates this issue by adapting the ImageNet-1K pre-trained network to multiple target datasets, however, this method is still expensive due to adapting procedure on each dataset.

Meta-learning Meta-learning (learning to learn) aims to train a model to generalize over a distribution of tasks, such that it can rapidly adapt to a new task (Vinyals et al., 2016; Snell et al., 2017; Finn et al., 2017; Nichol et al., 2018; Lee et al., 2019b; Hou et al., 2019). Recently, LEO (Rusu et al., 2019) proposed a scalable meta-learning framework which learns the latent generative representations of model parameters for a given data in a low-dimensional space for few-shot classification. Similarly to LEO (Rusu et al., 2019), our method learns a low-dimensional latent embedding space, but we learn a cross-modal space for both datasets and models for task-dependent model generation.

Neural Architecture Search with Meta-Learning Recent NAS methods with gradient-based meta-learning (Elsken et al., 2020; Lian et al., 2019; Shaw et al., 2019) have shown promising results on adapting to different tasks. However, they are only applicable on small scale tasks such as few-shot classification tasks (Elsken et al., 2020; Lian et al., 2019) and require high-computation time, due to the multiple unrolling gradient steps for one meta-update of each task. While some attempt to bypass the bottleneck with a first-order approximation (Lian et al., 2019; Shaw et al., 2019) or parallel computations with GPUs (Shaw et al., 2019), but their scalability is intrinsically limited due to gradient updates over a large number of tasks. To tackle such a scalability issue, we perform amortized inference over the multiple tasks by encoding a dataset into the low-dimensional latent vector and exploit fast GNN propagation instead of the expensive gradient update.

3 Method

Our goal is to output a high-performing neural architecture for a given dataset rapidly by learning the prior knowledge obtained from the rich database consisting of datasets and their corresponding

neural architectures. To this end, we propose Meta Dataset-to-Architecture (MetaD2A) framework which learns the cross-modal latent space of datasets and their neural architectures. Further, we introduce a meta-performance predictor, which predicts accuracies of given architectures without training the predictor on an unseen target dataset. Overview of the proposed approach is illustrated in Figure 1.

3.1 META-TRAINING NAS MODEL

To formally define the problem, let us assume that we have a source database of N_τ number of tasks, where each task $\tau = \{\mathcal{D}, \mathcal{G}, s\}$ consists of a dataset $\mathcal{D}$, a neural architecture represented as a Directed Acyclic Graph (DAG) $\mathcal{G}$ and an accuracy $\boldsymbol{s}$ obtained from the neural architecture $\mathcal{G}$ trained on $\mathcal{D}$. In the meta-training phase, the both dataset-to-architecture generator and meta-predictor learn to generalize over task distribution $p(\tau)$ using the source database. We describe how to empirically construct source database in Section 4.1.1.

3.1.1 LEARNING TO GENERATE GRAPHS FROM DATASETS

We propose a dataset-to-architecture generator which takes a dataset and then generates high-quality architecture candidates for the set. We want the generator to generate even novel architectures, which are not contained in the source database, at meta-test. Thus, the generator learns the *continuous* cross-modal latent space $\mathcal{Z}$ of datasets and neural architectures from the source database. For each task τ, the generator encodes dataset $\mathcal{D}$ as a vector $\mathbf{z}$ through the set encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})$ parameterized by $\boldsymbol{\phi}$ and then decodes a new graph $\tilde{\mathcal{G}}$ from $\mathbf{z}$ which are sampled from the prior $p(\mathbf{z})$ by using the graph decoder $p_{\boldsymbol{\theta}}(\mathcal{G}|\mathbf{z})$ parameterized by $\boldsymbol{\theta}$. Then, our goal is that $\tilde{\mathcal{G}}$ generated from $\mathcal{D}$ to be the true $\mathcal{G}$ which is pair of $\mathcal{D}$. We meta-learn the generator using set-amortized inference, by maximizing the approximated evidence lower bound (ELBO) as follows:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \sum_{\tau \sim p(\tau)} \mathcal{L}^{\tau}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathcal{D}, \mathcal{G}) \tag{1}$$

where

$$\mathcal{L}^{\tau}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})}\big[\log p_{\boldsymbol{\theta}}(\mathcal{G}|\mathbf{z})\big] - \lambda \cdot L^{\tau}_{KL}\big[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})||p(\mathbf{z})\big] \tag{2}$$

Each dimension of the prior $p(\mathbf{z})$ factorizes into $\mathcal{N}(0, 1)$. L^{τ}_{KL} is the KL divergence between two multivariate Gaussian distributions which has a simple closed form (Kingma & Welling, 2014) and λ is the scalar weighting value. Using the reparameterization trick on $\mathbf{z}$, we optimize the above objective by stochastic gradient variational Bayes (Kingma & Welling, 2014). We use a set encoder described in Section 3.1.3 and we adopt a Graph Neural Network (GNN)-based decoder for directed acyclic graph (DAG)s (Zhang et al., 2019), which allows message passing to happen only along the topological order of the DAGs. For detailed descriptions for the generator, see Section A of Suppl.

3.1.2 META-PERFORMANCE PREDICTOR

While many performance predictor for NAS have been proposed (Luo et al., 2018; Cai et al., 2020; Lu et al., 2020; Zhou et al., 2020; Tang et al., 2020), those performance predictors repeatedly collect architecture-accuracy database for each new dataset, which results in huge total cost on many datasets. Thus, the proposed predictor $f_{\boldsymbol{\omega}}(s|\mathcal{D}, \mathcal{G})$ takes a **dataset** as well as graph as an input to support multiple datasets, while the existing performance predictor takes a graph only. Then, the proposed predictor meta-learns set-dependent performance proxy generalized over the task distribution $p(\tau)$ in the meta-training stage. This allows the meta-learned predictor to accurately predict performance on unseen datasets without additional training. The proposed predictor $f_{\boldsymbol{\omega}}$ consists of a dataset encoder and a graph encoder, followed by two linear layers with $relu$. For dataset encoding, we use the set encoder of Section 3.1.3 which takes $\mathcal{D}$ as an input. We adopt direct acyclic graph encoder (Zhang et al., 2019) for DAG $\mathcal{G}$ (Please refer to Section B of Suppl.). We concatenate the outputs of both graph encoder and the set encoder, and feed them to two linear layers with $relu$ to predict accuracy. We train the predictor $f_{\boldsymbol{\omega}}$ to minimize the MSE loss $\mathcal{L}^{\tau}_{\boldsymbol{\omega}}(s, \mathcal{D}, \mathcal{G})$ between the predicted accuracy and the true accuracy s of the model on each task sampled from the source database:

$$\min_{\boldsymbol{\omega}} \sum_{\tau \sim p(\tau)} \mathcal{L}^{\tau}_{\boldsymbol{\omega}}(s, \mathcal{D}, \mathcal{G}) = \sum_{\tau \sim p(\tau)} (s - f_{\boldsymbol{\omega}}(\mathcal{D}, \mathcal{G}))^2 \tag{3}$$

3.1.3 SET ENCODER

The efficacy of the proposed framework is dependent on how accurately set encoder captures the distribution of the target dataset and extracts information related with the goal of the generator and the predictor. To compress the entire instances from a dataset $\mathcal{D}$ into a single latent code $\mathbf{z}$, the set encoder should process input sets of any size and summarize consistent information agnostically to the order of the instances (permutation-invariance). Existing set encoders such as *DeepSet* (Zaheer et al., 2017), *SetTransformer* (Lee et al., 2019a), and *StatisticsPooling* (Lee et al., 2020) fulfill those requirements and might be used. However, *DeepSet* and *SetTransformer* are non-hierarchical poolings, thus cannot accurately model individual classes in the given dataset. Moreover, *DeepSet* and *StatisticsPooling* resort to simple averaging of the instance-wise representations.

Therefore, we introduce a novel set encoder which stacks two permutation-invariant modules with attention-based learnable parameters. The lower-level intra-class encoder captures the class prototypes that reflect label information, and the high-level inter-class encoder considers the relationship between class prototypes and aggregates them into a latent vector. The proposed structure of the set encoder models high-order interactions between the set elements allowing the generator and predictor to effectively extract useful information to achieve each goal.

Specifically, for a given dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{Y}\}$, where $\boldsymbol{X} = \{\boldsymbol{X}_c\}_{c=1}^{C}$ and $\boldsymbol{Y} = \{\boldsymbol{Y}_c\}_{c=1}^{C}$ are the set of instances and target labels of C classes respectively. We randomly sample instances $\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c\} \in \mathbb{R}^{\mathrm{b_c} \times \mathrm{d_x}}$ of class c, where $\mathbf{x}$ is a $\mathrm{d_x}$ dimensional feature vector, $\boldsymbol{B}_c \subset \boldsymbol{X}_c$ and $||\boldsymbol{B}_c|| = \mathrm{b_c}$. We input the sampled instances into the IntraSetPool, the intra-class encoder, to encode class prototype $\mathbf{v}_c \in \mathbb{R}^{1 \times \mathrm{d_{\mathbf{v}_c}}}$ for each class $c = 1, ..., C$. Then we further feed the class-specific set representations $\{\mathbf{v}_c\}_{c=1}^{C}$ into the InterSetPool, the inter-class encoder, to generate the dataset representation $\mathbf{h}_e \in \mathbb{R}^{1 \times \mathrm{d_{\mathbf{h}_e}}}$ as follows:

$$\mathbf{v}_c = \text{IntraSetPool}\big(\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c\}\big), \quad \mathbf{h}_e = \text{InterSetPool}\big(\{\mathbf{v}_c\}_{c=1}^{C}\big) \tag{4}$$

Both the set poolings are stacked attention-based blocks borrowed from Lee et al. (2019a). Note that while Lee et al. (2019a) is an attention-based set encoder, it ignores class label information of given dataset, which may lead to poor performance. Please see Section C of the Suppl. for more details.

3.2 META-TEST (SEARCHING)

In the meta-test stage, for an unseen dataset $\hat{\mathcal{D}}$, we can obtain n set-dependent DAGs $\{\hat{\mathcal{G}}_i\}_{i=1}^{n}$, with the meta-trained generator parameterized by $\boldsymbol{\phi}^*$ and $\boldsymbol{\theta}^*$, by feeding $\hat{\mathcal{D}}$ as an input. Through such set-level amortized inference, our method can easily generate neural architecture(s) for the novel dataset. The latent code $\mathbf{z} \in \mathbb{R}^{1 \times \mathrm{d_z}}$ can be sampled from a dataset-conditioned Gaussian distribution with diagonal covariance where $\text{NN}_{\boldsymbol{\mu}}, \text{NN}_{\boldsymbol{\sigma}}$ are single linear layers:

$$\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) \quad \text{where} \quad \boldsymbol{\mu}, \boldsymbol{\sigma} = \text{NN}_{\boldsymbol{\mu}}(\mathbf{h}_e), \text{NN}_{\boldsymbol{\sigma}}(\mathbf{h}_e) \tag{5}$$

In the meta-test, the predictor $f_{\boldsymbol{\omega}^*}(\hat{s}_i|\hat{\mathcal{D}}, \hat{\mathcal{G}}_i)$ predicts accuracies $\{\hat{s}_i\}_{i=1}^{n}$ for a given unseen dataset $\hat{\mathcal{D}}$ and each generated architecture of $\{\hat{\mathcal{G}}_i\}_{i=1}^{n}$ and then select the neural architecture having the highest predicted accuracy among $\{\hat{s}_i\}_{i=1}^{n}$.

4 EXPERIMENT

We conduct extensive experiments to validate MetaD2A framework. First, we compare our model with conventional NAS methods on NAS-Bench-201 search space in Section 4.1. Second, we compare our model with transferable NAS method under a large search space in Section 4.2. Third, we compare our model with other Meta-NAS approaches on few-shot classification tasks in Section 4.3. Finally, we analyze the effectiveness of our framework in Section 4.4.

4.1 NAS-BENCH-201 SEARCH SPACE

4.1.1 EXPERIMENT SETUP

We learn our model on source database consisting of subsets of ImageNet-1K and neural architectures of NAS-Bench-201 (Dong & Yang, 2020) and (meta-)test our model by searching for architectures on 6 benchmark datasets without additional NAS model training.

NAS-Bench-201 search space contains cell-based neural architectures, where each cell is represented as directed ayclic graph (DAG) consisting of the 4 nodes and the 6 edge connections. For each edge connection, NAS models select one of 5 operation candidates such as zerorize, skip connection, 1-by-1 convolution, 3-by-3 convolution, and 3-by-3 average pooling.

Source Database To meta-learn our model, we practically collect multiple tasks where each task consists of (dataset, architecture, accuracy). We compile ImageNet-1K (Deng et al., 2009) as multiple sub-sets by randomly sampling 20 classes with an average of 26K images for each sub-sets and assign them to each task. All images are downsampled by 32×32 size. We search for the set-specific architecture of each sampled dataset using random search among high-quality architectures which are included top-5000 performance architecture group on ImageNet-16-120 or GDAS (Dong & Yang, 2019b). For the predictor, we additionally collect 2,920 tasks through random sampling. We obtain its accuracy by training the architecture on dataset of each task. We collect N_τ =1,310/4,230 meta-training tasks for the generator/predictor and 400/400 meta-validation tasks for them, respectively. Meta-training time is 12.7/8.4 GPU hours for the generator/the predictor and note that meta-training phase is needed only once for all experiments of NAS-Bench-201 search space.

Meta-Test Datasets We apply our model trained from source database to 6 benchmark datasets such as 1) CIFAR-10 (Krizhevsky et al., 2009), 2) CIFAR-100 (Krizhevsky et al., 2009), 3) MNIST (LeCun & Cortes, 2010), 4) SVHN (Netzer et al., 2011), 5) Aircraft (Maji et al., 2013), and 6) Oxford-IIIT Pets (Parkhi et al., 2012). On CIFAR10 and CIFAR100, the generator generates 500 neural architectures and we select 30 architectures based on accuracies predicted by the predictor. Following SETN (Dong & Yang, 2019a), we retrieve the accuracies of N architecture candidates from the NAS-bench-201 and report the highest final accuracy for each run. While $N = 1000$ in SETN, we set a smaller number of samples ($N = 30$) for MetaD2A. We report the mean accuracies over 10 runs of the search process by retrieving accuracies of searched architectures from NAS-Bench-201. On MNIST, SVHN, Aircraft, and Oxford-IIIT Pets, the generator generates 50 architectures and select the best one with the highest predicted accuracy. we report the accuracy averaging over 3 runs with different seeds. For fair comparison, the searched architectures from our model are trained on each target datasets from the scratch. Note that once trained MetaD2A can be used for more datasets without additional training. Our model is performed with a single Nvidia 2080ti GPU.

4.1.2 Results on Unseen Datasets

Table 1 shows that our model meta-learned on the source database can successfully generalize to 6 unseen datasets such as MNIST, SVHN, CIFAR-10, CIFAR-100, Aircraft, and Oxford-IIIT Pets by outperforming all baselines. Since the meta-learned MetaD2A can output set-specialized architectures on target datasets through inference process with **no** training cost, the search speed is extremely fast. As shown in Table 1, the search time of MetaD2A averaging on 6 benchmark datasets is within 33 GPU second. This is impressive results in that it is at least 147× (maximum: 12169×) faster than conventional set-specific NAS approaches which need training NAS models on each target dataset. Such rapid search of REA, RS, REINFORCE and BOHB is only possible where all of the accuracies are pre-computed like NAS-Bench201 so that it can retrieve instantly on the target dataset, therefore, it is difficult to apply them to other non-benchmark datasets. Especially, we observe that MetaD2A which is learned over multiple tasks benefit to search set-dependent neural architectures for fine-grained datasets such as Aircraft and Oxford-IIIT Pets.

4.2 MobileNetV3 Search Space

4.2.1 Experiment Setup

We apply our meta-trained model on four unseen datasets, comparing with transferable NAS (NSGANetV2 (Lu et al., 2020)) under the same search space of MobileNetV3, where it contains more than 10^{19} architectures. Each CNN architecture consists of five sequential blocks and the targets of searching are the number of layers, the number of channels, kernel size, and input resolutions. For a fair comparison, we also exploit the supernet for the parameters as NSGANetV2 does. We collect $N_\tau = 3,018/153,408$ meta-training tasks for the generator/predictor and $646/32,872$ meta-validation tasks, respectively as a source database from the ImageNet-1K dataset and architectures of MobileNetV3 search space. Meta-training time is $2.21/1.41$ GPU days for the generator/the predictor. Note that the meta-training phase is needed only **once** on the source database.

Table 1: **Performance on Unseen Datasets (Meta-Test)** MetaD2A conducts amortized inference on unseen target datasets after meta-training on source database consisting of subsets of ImageNet-1K and architectures of NAS-Bench-201 search space. Meta-training time is 12.7/8.4 GPU hours for the generator/the predictor. For fair comparison, the parameters of searched architectures are trained on each dataset from scratch instead of transferring parameters from ImageNet. T is the time to construct precomputed architecture database for each target. We report accuracies with 95% confidence intervals.

Target Dataset	NAS Method	NAS Training-free on Target	Params (M)	Search Time (GPU Sec)	Speed Up	Search Cost ($)	Accuracy (%)
CIFAR-10	ResNet (He et al., 2016)		0.86	N/A	N/A	N/A	93.97±0.00
	REA (Real et al., 2019)		-	0.02+T	-	-	93.92±0.30
	RS (Bergstra & Bengio, 2012)		-	0.01+T	-	-	93.70±0.36
	REINFORCE (Williams, 1992)		-	0.12+T	-	-	93.85±0.37
	BOHB (Falkner et al., 2018)		-	3.59+T	-	-	93.61±0.52
	RSPS (Li & Talwalkar, 2019)		-	10200	147×	4.13	84.07±3.61
	SETN (Dong & Yang, 2019a)		-	30200	437×	12.25	87.64±0.00
	GDAS (Dong & Yang, 2019b)		-	25077	363×	10.17	93.61±0.09
	PC-DARTS (Xu et al., 2020)		1.17	10395	150×	4.21	93.66±0.17
	DrNAS (Chen et al., 2021)		1.53	21760	315×	8.82	94.36±0.00
	MetaD2A (Ours)	✓	1.11	**69**	1×	**0.028**	**94.37±0.03**
CIFAR-100	ResNet (He et al., 2016)		0.86	N/A	N/A	N/A	70.86±0.00
	REA (Real et al., 2019)		-	0.02+T	-	-	71.84±0.99
	RS (Bergstra & Bengio, 2012)		-	0.01+T	-	-	71.04±1.07
	REINFORCE (Williams, 1992)		-	0.12+T	-	-	71.71±1.09
	BOHB (Falkner et al., 2018)		-	3.59+T	-	-	70.85±1.28
	RSPS (Li & Talwalkar, 2019)		-	18841	196×	7.64	52.31±5.77
	SETN (Dong & Yang, 2019a)		-	58808	612×	23.85	59.09±0.24
	GDAS (Dong & Yang, 2019b)		-	51580	537×	20.91	70.70±0.30
	PC-DARTS (Xu et al., 2020)		0.26	19951	207×	8.09	66.64±2.34
	DrNAS (Chen et al., 2021)		1.20	34529	359×	14.00	**73.51±0.00**
	MetaD2A (Ours)	✓	1.07	**96**	1×	**0.039**	**73.51±0.00**
MNIST	ResNet (He et al., 2016)		0.86	N/A	N/A	N/A	99.67±0.01
	RSPS (Li & Talwalkar, 2019)		0.25	22457	3208×	9.10	99.63±0.02
	SETN (Dong & Yang, 2019a)		0.56	69656	9950×	28.24	99.69±0.04
	GDAS (Dong & Yang, 2019b)		0.82	60186	8598×	24.40	99.64±0.04
	PC-DARTS (Xu et al., 2020)		0.62	24857	3551×	10.08	99.66±0.04
	DrNAS (Chen et al., 2021)		1.53	44131	6304×	17.89	99.59±0.02
	MetaD2A (Ours)	✓	0.61	**7**	1×	**0.002**	**99.71±0.08**
SVHN	ResNet (He et al., 2016)		0.86	N/A	N/A	N/A	96.13±0.19
	RSPS (Li & Talwalkar, 2019)		0.48	27962	3994×	11.34	96.17±0.12
	SETN (Dong & Yang, 2019a)		0.48	85189	12169×	34.54	96.02±0.12
	GDAS (Dong & Yang, 2019b)		0.24	71595	10227×	10.17	95.57±0.57
	PC-DARTS (Xu et al., 2020)		0.47	31124	4446×	12.62	95.40±0.67
	DrNAS (Chen et al., 2021)		1.53	52791	7541×	21.40	96.30±0.05
	MetaD2A (Ours)	✓	0.86	**7**	1×	**0.004**	**96.34±0.37**
Aircraft	ResNet (He et al., 2016)		0.86	N/A	N/A	N/A	47.01±1.16
	RSPS (Li & Talwalkar, 2019)		0.22	18697	1869×	7.58	42.19±3.88
	SETN (Dong & Yang, 2019a)		0.44	18564	1856×	7.52	44.84±3.96
	GDAS (Dong & Yang, 2019b)		0.62	18508	1850×	7.50	53.52±0.48
	PC-DARTS (Xu et al., 2020)		0.32	3524	352×	1.42	26.33±3.40
	DrNAS (Chen et al., 2021)		1.03	34529	3452×	13.14	46.08±7.00
	MetaD2A (Ours)	✓	0.83	**10**	1×	**0.004**	**58.43±1.18**
Oxford-IIIT Pets	ResNet (He et al., 2016)		0.86	N/A	N/A	N/A	25.58±3.43
	RSPS (Li & Talwalkar, 2019)		0.32	3360	420×	1.36	22.91±1.65
	SETN (Dong & Yang, 2019a)		0.32	8625	1078×	3.49	25.17±1.68
	GDAS (Dong & Yang, 2019b)		0.83	6965	870×	2.82	24.02±2.75
	PC-DARTS (Xu et al., 2020)		0.44	2844	355×	1.15	25.31±1.38
	DrNAS (Chen et al., 2021)		0.44	6019	752×	2.44	26.73±2.61
	MetaD2A (Ours)	✓	0.83	**8**	1×	**0.003**	**41.50±4.39**

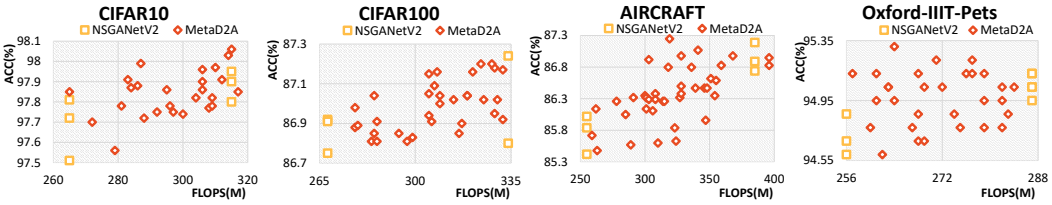

Figure 3: **Performance on Unseen Datasets (Meta-Test)** We show accuracy over flop of both MetaD2A and a transferable NAS referred as to NSGANetV2 (Lu et al., 2020) after meta-training MetaD2A on source database consisting of subsets of ImageNet-1K and architectures in MobileNetV3 search space. Note that each plot point is searched within 125 GPU seconds by MetaD2A.

4.2.2 RESULTS ON UNSEEN DATASETS

We search and evaluate the architecture multiple times with both NSGANetV2 and ours on four unseen datasets such as CIFAR-10, CIFAR-100, Aircraft, and Oxford-IIIT Pets with different random seeds. Search times of MetaD2A for CIFAR-10, CIFAR-100, Aircraft, and Oxford-IIIT Pets are within 57, 195, 77, and 170 GPU seconds on average with a single Nvidia RTX 2080ti GPU respectively, while NSGANetV2 needs 1 GPU day with 8 1080ti GPUs on each dataset, which is 5,523 times slower than MetaD2A. Besides the huge speed up, Figure 3 shows that our model can search for a comparable architecture to the NSGANetV2 over flops without a performance drop. Interestingly, even we use naive flop filtering and NSGANetV2 uses an objective function for flop constraints, MetaD2A performs consistently comparably to NSGANetV2 over the different flops. Overall, the results demonstrate that our model also can generalize to unseen datasets not only under the NAS-Bench-201 space but also under a larger MobileNetV3 space with its meta-knowledge.

4.3 COMPARISON WITH META-NAS APPROACHES

We further compare our method against Meta-NAS methods (Kim et al., 2018; Elsken et al., 2020; Lian et al., 2019; Shaw et al., 2019) on few-shot classification tasks, which are the main setting existing Meta-NAS methods have been consider. Following (Elsken et al., 2020; Lian et al., 2019), we adopt bi-level optimization (e.g., MAML framework) to meta-learn initial weights of neural architectures searched by our model on a meta-training set of mini-imagenet. As shown in the Table 2, the few-shot classification results on MiniImageNet further clearly show the MetaD2A's effectiveness over existing Meta-NAS methods, as well as the conventional meta-learning methods without NAS (Finn et al., 2017; Antoniou et al., 2018).

Method	NAS	Params (K)	MiniImageNet 5way 1shot	MiniImageNet 5way 5shot
MAML (Finn et al., 2017)		32.9	48.70	63.11
MAML++ (Antoniou et al., 2018)		32.9	52.15	68.32
AutoMeta (Kim et al., 2018)	✓	28	49.58	65.09
BASE (Shaw et al., 2019)	✓	1200	-	66.20
T-NAS++ (Lian et al., 2019)	✓	26.5	54.11	69.59
MetaNAS (Elsken et al., 2020)	✓	30	49.7	62.1
MetaD2A (Ours)	✓	28.9	**54.71**	**70.59**

Table 2: **Performance on Few-shot Classification Task**

4.4 EFFECTIVENESS OF METAD2A

Now, we verify the efficacy of each component of MetaD2A with further analysis.

Ablation Study on MetaD2A We train different variations of our model on the subsets of ImageNet-1K, and test on CIFAR10, CIFAR100, and Aircraft in Table 3 with the same experimental setup as the main experiments in Table 1. The MetaD2A generator without the performance predictor (Generator only) outperforms the simple random architecture sampler (Random Sampling), especially by 15.3% on Aircraft, which demonstrates the effectiveness of MetaD2A over the random sampler. Also, we observe that combining the meta-performance predictor to the random architecture sampler (Predictor only) enhances the accuracy of the final architecture on all datasets. Finally, MetaD2A combined with the performance predictor (MetaD2A) outperforms all baselines, especially by 20.28% on Aircraft, suggesting that our MetaD2A can output architectures that are more relevant to the given task.

Model	G	P	Target Dataset CIFAR10	CIFAR100	Aircraft
Random Sampling			93.06$\pm$0.55	69.94$\pm$1.21	38.15$\pm$0.99
Generator only	✓		93.96$\pm$0.22	71.54$\pm$0.63	53.45$\pm$3.27
Predictor only		✓	93.70$\pm$0.32	72.33$\pm$0.88	53.39$\pm$3.13
MetaD2A	✓	✓	**94.37**$\pm$**0.03**	**73.51**$\pm$**0.00**	**58.43**$\pm$**1.18**

Table 3: **Ablation Study of MetaD2A on Unseen Datasets**

Effectiveness of Set-to-Architecture Generator

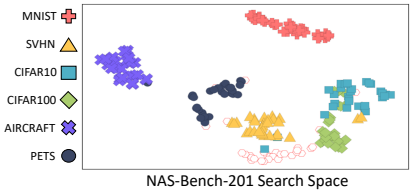

Figure 4: **T-SNE vis. of Latent Space**

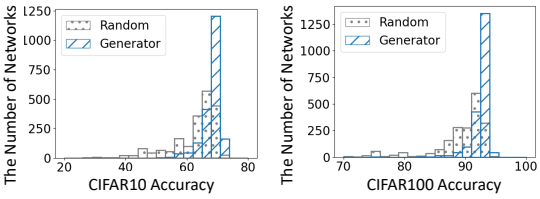

Figure 5: **The Quality of Generated Architectures**

We first visualize cross-modal latent embeddings $\{\mathbf{z}\}$ of unseen datasets encoded by the meta-

learned generator with T-SNE in Figure 4. Each marker indicates $\{\mathbf{z}\}$ of the sampled subsets of each dataset with different seeds. We observe that the generator classifies well embeddings $\{\mathbf{z}\}$ by datasets in the latent space while clusters $\mathbf{z}$ of the subset of the same dataset. Furthermore, we investigate the quality of generated architectures from those embeddings $\{\mathbf{z}\}$. In the Figure 5, the generator sample 2000 architecture candidates from the embeddings encoded each target dataset and computes the validate accuracy of those architectures. The proposed generator generates more high-performing architectures than the simple random architecture sampler for each target dataset. These results are consistent with Table 3, where the generator (Generator only) enhances the performance compared with the simple random architecture sampler (Random Sampling) consistently on CIFAR10 and CIFAR100. The meta-learned generator allows us to effective and efficient search by excluding the poor-performing architectures of broad search space. We believe the generator replaces the random sampling stage of other NAS methods. We leave to valid it as the future work.

Could the Generator Create Novel Architectures? Since the generator maps set-architecture pairs in the **continuous** latent space, it can generate novel architectures in the meta-test, which are not contained in the source database. To validate it, we evaluate generated 10,000 neural architecture samples of both search space with the measures *Validity*, *Uniqueness*, and *Novelty* following (Zhang et al., 2019) in Table 4. Each is defined as how often the model can generate valid neural architectures from the prior distribution, the proportion of unique graphs out of the valid generations, and the proportion of valid generations that are not included in the training set, respectively. For NAS-Bench-201 search space and MobileNetV3 search space, respectively, the results show the meta-learned generator can generate 67.31%/100% new graphs that do not belong to the training set and can generate 35.19%/100% various graphs, not picking always-the-same architecture seen of the source database.

Search Space	Validity	Uniqueness	Novelty
NAS-Bench-201	1.0000	0.3519	0.6731
MobileNetV3	0.9831	1.0000	1.0000

Table 4: **Analysis of Generated Architectures**

Effectiveness of Meta-Predictor We first demonstrate the necessity of set encoding to handle multiple datasets with a single predictor. In Table 5, we meta-train all models on the source database of NAS-Bench-201 search space and measure Pearson correlation coefficient on the validation tasks (400 unseen tasks) of the source database. Pearson correlation coefficient is the linear correlation between the actual performance and the predicted performance (higher the better). Using both the dataset and the computational graph of the target architecture as inputs, instead of using graphs only (Graph Encoder Only), clearly leads to better performance to support multiple datasets. Moreover, the predictor with the proposed set encoder clearly shows a higher correlation than other set encoders (DeepSet (Zaheer et al., 2017), SetTransformer (Lee et al., 2019a), and Statistical Pooling (Lee et al., 2020)).

Predictor Model	Input Type Data	Input Type Graph	Pearson Corr. Coeff.
Graph Encoder (GE) Only		✓	0.6439
DeepSet (Zaheer et al., 2017) + GE	✓	✓	0.7286
SetTransformer (Lee et al., 2019a) + GE	✓	✓	0.7744
Statistical Pooling (Lee et al., 2020) + GE	✓	✓	0.7796
The Proposed Set Encoder + GE (Ours)	✓	✓	**0.8085**

Table 5: **Effectiveness of Set Encoding to Accurately Predict the Accuracy of Multiple Datasets**

5 Conclusion

We proposed a novel NAS framework, MetaD2A (Meta Dataset-to-Architecture), that can output a neural architecture for an unseen dataset. The MetaD2A generator learns a dataset-to-architecture transformation over a database of datasets and neural architectures by encoding each dataset using a set encoder and generating each neural architecture with a graph decoder. While the model can generate a novel architecture given a new dataset in an amortized inference, we further learn a meta-performance predictor to select the best architecture for the dataset among multiple sampled architectures. The experimental results show that our method shows competitive performance with conventional NAS methods on various datasets with very small search time as it generalizes well across datasets. We believe that our work is a meaningful step towards building a practical NAS system for real-world scenarios, where we need to handle diverse datasets while minimizing the search cost.

Acknowledgements This work was supported by Samsung Research Funding Center of Samsung Electronics (No. SRFC-IT1502-51) and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST)).

REFERENCES

Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

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

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *In International Conference on Learning Representations (ICLR)*, 2017.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*, 2019.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Dr{nas}: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021.

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. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, 2019a.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition (CVPR)*, 2019b.

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

Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2020.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

Jiemin Fang, Yuzhu Sun, Kangjian Peng, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Fast neural network adaptation via parameter remapping and architecture search. *arXiv preprint arXiv:2001.02525*, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

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 (CVPR)*, 2016.

Ruibing Hou, Hong Chang, MA Bingpeng, Shiguang Shan, and Xilin Chen. Cross attention network for few-shot classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *International Conference on Machine Learning (ICML)*, 2018.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in neural information processing systems (NeurIPS)*, 2018.

Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Youngduck Choi, Yongseok Choi, Dong-Yeon Cho, and Jiwon Kim. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Hae Beom Lee, Hayeon Lee, Donghyun Na, Saehoon Kim, Minseop Park, Eunho Yang, and Sung Ju Hwang. Learning to balance: Bayesian meta-learning for imbalanced and out-of-distribution tasks. In *International Conference on Learning Representations (ICLR)*, 2020.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning (ICML)*, 2019a.

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019b.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pp. 367–377. PMLR, 2019.

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 *International Conference on Learning Representations (ICLR)*, 2019.

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)*, 2018.

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

Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European Conference on Computer Vision*, pp. 35–51. Springer, 2020.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems (NeurIPS)*, 2018.

Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, 2018.

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 (AAAI)*, 2019.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. 2019.

Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems (NIPS)*, 2017.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich.

Yehui Tang, Yunhe Wang, Yixing Xu, Hanting Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. A semi-supervised assessor of neural architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1810–1819, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems (NIPS)*, 2016.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

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

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

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11396–11404, 2020.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *In International Conference on Learning Representations (ICLR)*, 2017.

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 (CVPR)*, 2018.

A DETAILS OF THE GENERATOR

A.1 GRAPH DECODING

To generate the ith node v_i, we compute the operation type $\mathrm{o}_{v_i} \in \mathbb{R}^{1\times n_o}$ over n_o operations based on the current graph state $\mathbf{h}_\mathcal{G} := \mathbf{h}_{v_{i-1}}$ and then predict whether the edge exists between the node v_i and other existing nodes. Following (Zhang et al., 2019), when we compute the edge probability $e_{\{v_j,v_i\}}$, we consider nodes $\{v_j | j = i-1, ..., 1\}$ in the reverse order to reflect information from nodes close to v_i to the root node when deciding whether edge connection. Note that the proposed process guarantees the generation of directed acyclic graph since directed edge is always created from existing nodes to a new node.

The graph decoder starts from an initial hidden state $\mathbf{h}_{v_0} = \mathrm{NN}_{\mathrm{init}}(\mathbf{z})$, where $\mathrm{NN}_{\mathrm{init}}$ is an MLP followed by $tanh$. For i^{th} node v_i according to *topological order*, we compute the probability of each operation type $\mathrm{o}_{v_i} \in \mathbb{R}^{1\times n_o}$ over n_o operations, given the current graph state as the last hidden node $\mathbf{h}_\mathcal{G} := \mathbf{h}_{v_i}$. That is, $\mathrm{o}_{v_i} = \mathrm{NN}_{\mathrm{node}}(\mathbf{h}_\mathcal{G})$, where $\mathrm{NN}_{\mathrm{node}}$ is an MLP followed by *softmax*. When the predicted v_i type is the end-of-graph, we stop the decoding process and connect all leaf nodes to v_i. Otherwise we update hidden state $\mathbf{h}_{v_i}^{(t)}$ at time step t as follows:

$$\begin{aligned} \mathbf{h}_{v_i}^{(t+1)} &= \mathrm{UPDATE}(i, \boldsymbol{m}_{v_i}^{(t)}) \\ \text{where } \boldsymbol{m}_{v_i}^{(t)} &= \sum_{u\in\mathcal{V}_{v_i}^{in}} \mathrm{AGGREGATE}(\mathbf{h}_u^{(t)}) \end{aligned} \tag{6}$$

The function UPDATE is a gated recurrent unit (GRU) (Cho et al., 2014), i is the order of v_i, and $\boldsymbol{m}_{v_i}^{(t)}$ is the incoming message to v_i. The function AGGREGATE consists of mapping and gating functions with MLPs, where $\mathcal{V}_{v_i}^{in}$ is a set of predecessors with incoming edges to v_i. For all previously processed nodes $\{v_j | j = i-1, ..., 1\}$, we decide whether to link an edge from v_j to v_i by sampling the edge based on edge connection probability $e_{\{v_j,v_i\}} = \mathrm{NN}_{\mathrm{edge}}(\mathbf{h}_j, \mathbf{h}_i)$, where $\mathrm{NN}_{\mathrm{edge}}$ is a MLP followed by $sigmoid$. We update $\mathbf{h}_{v_i}$ by Eq. (6) whenever a new edge is connected to v_i. For meta-test, we select the operation with the max probability for each node and edges with $e_{\{v_j,v_i\}} > 0.5$.

A.2 META-TRAINING OBJECTIVE

We meta-learn the model using Eq. (1). The expectation of the log-likelihood $\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})}\big[\log p_{\boldsymbol{\theta}}(\mathcal{G}|\mathbf{z})\big]$ of (2) can be rewritten with negative cross-entropy loss $-L_{CE}^{\tau}$ for nodes and binary cross-entropy loss $-L_{BCE}^{\tau}$ for edges, and we slightly modify it using the generated set-dependent graph $\tilde{\mathcal{G}}$ and the ground truth graph $\mathcal{G}$ as the input as follows:

$$-\sum_{i\in\mathcal{V}} \Big\{ L_{CE}^{\tau}(\tilde{o}_i, o_i) + \sum_{j\in\mathcal{V}_i} L_{BCE}^{\tau}(\tilde{e}_{\{j,i\}}, e_{\{j,i\}}) \Big\} \tag{7}$$

We substitute the log-likelihood term of Eq. (2) such as Eq. (7) and learn the proposed generator by maximizing the objective (1) to learn $\boldsymbol{\phi}, \boldsymbol{\theta}$, which are shared across all tasks.

B GRAPH ENCODING OF THE SET-DEPENDENT PREDICTOR

For a given graph candidate $\mathcal{G}$, we sequentially perform message passing for nodes from the predecessors following the *topological order* of the DAG $\mathcal{G}$. We iteratively update hidden states $\mathbf{h}_{v_i}^{(t)}$ using the Eq. (8) by feeding in its predecessors' hidden states $\{u \in \mathcal{V}_{v_i}^{in}\}$.

$$\begin{aligned} \mathbf{h}_{v_i}^{(t+1)} &= \mathrm{UPDATE}(\boldsymbol{y}_{v_i}, \boldsymbol{m}_{v_i}^{(t)}) \\ \text{where } \boldsymbol{m}_{v_i}^{(t)} &= \sum_{u\in\mathcal{V}_{v_i}^{in}} \mathrm{AGGREGATE}(\mathbf{h}_u^{(t)}) \end{aligned} \tag{8}$$

For starting node v_0 which the set of predecessors is the empty, we output the zero vector as the hidden state of v_0. We use the last hidden states of the ending node as the output of the graph

encoder $\mathbf{h}_f$. Additionally, we exploit Bi-directional encoding (Zhang et al., 2019) which reverses the node orders to perform the encoding process. In this case, the final node becomes the starting point. Thus, the backward graph encoder outputs $\mathbf{h}_b$, which is the last hidden states of the starting node. We concatenate the outputs $\mathbf{h}_f$ of the forward graph encoder and $\mathbf{h}_b$ of the backward graph encoder as the final output of the Bi-directional graph encoding.

C Building Blocks of the Set Encoder

We use Set Attention Block (SAB) and Pooling by Multi-head Attention (PMA) (Lee et al., 2019a), where the former learns the features for each element in the set using self-attention while the latter pools the input features into k representative vectors. Set Attention Block (SAB) is an attention-based block, which makes the features of all of the instances in the set reflect the relations between itself and others such as:

$$\begin{aligned} \mathrm{SAB}(\boldsymbol{X}) &= \mathrm{LN}(\boldsymbol{H} + \mathrm{MLP}(\boldsymbol{H})) \\ \text{where} \quad \boldsymbol{H} &= \mathrm{LN}(\boldsymbol{X} + \mathrm{MH}(\boldsymbol{X}, \boldsymbol{X}, \boldsymbol{X})) \end{aligned} \tag{9}$$

where LN and MLP is the layer normalization (Ba et al., 2016) and the multilayer perceptron respectively, and $\boldsymbol{H} \in \mathbb{R}^{n_{\boldsymbol{B}_c} \times \mathrm{d}_{\boldsymbol{H}}}$ is computed with multi-head attention $\mathrm{MH}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$ (Vaswani et al., 2017) which queries, keys, and values are elements of input set $\boldsymbol{X}$.

Features encoded from the SAB layers can be pooled by PMA on learnable seed vectors $\boldsymbol{S} \in \mathbb{R}^{k \times \mathrm{d}_{\boldsymbol{S}}}$ to produce k vectors by slightly modifying $\boldsymbol{H}$ calculation of Eq. (9):

$$\begin{aligned} \mathrm{PMA}(\boldsymbol{X}) &= \mathrm{LN}(\boldsymbol{H} + \mathrm{MLP}(\boldsymbol{H})) \\ \text{where} \quad \boldsymbol{H} &= \mathrm{LN}(\boldsymbol{X} + \mathrm{MH}(\boldsymbol{S}, \mathrm{MLP}(\boldsymbol{X}), \mathrm{MLP}(\boldsymbol{X}))) \end{aligned} \tag{10}$$

While k can be any size (i.e. k=1,2,10,16), we set $k = 1$ for generating the single latent vector. For extracting consistent information not depending the order and the size of input elements, encoding functions should be constructed by stacking *permutation-equivariant* layers E, which satisfies below condition for any permutation π on a set $\boldsymbol{X}$ (Zaheer et al., 2017):

$$\mathrm{E}(\{\mathrm{x} | \mathrm{x} \in \pi \boldsymbol{X}\}) = \pi \mathrm{E}(\{\mathrm{x} | \mathrm{x} \in \boldsymbol{X}\}) \tag{11}$$

Since all of the components in SAB and PMA are row-wise computation functions, SAB and PMA is permutation equivarint by definition Eq. (11).

D Search Space

Following the NAS-Bench-201 (Dong & Yang, 2020), We explore the search space consisting of 15,625 possible cell-based neural architectures for all experiments. Macro skeleton is stacked with one stem cell, three stages consisting of 5 cells for each, and a residual block (He et al., 2016) between stages. The stem cell consists of 3-by-3 convolution with 16 channels and cells of the first, second and third stages have 16, 32 and 64, respectively. Residual blocks have convolution layer with the stride 2 for down-sampling. A fully connected layer is attached to the macro skeleton for classification. Each cell is DAG which consists of the fixed 4 nodes and the fixed 6 edge connections. For each edge connection, NAS models select one of 5 operation candidates such as zerorize, skip connection, 1-by-1 convolution, 3-by-3 convolution, and 3-by-3 average pooling. To effectively encode the operation information as the node features, we represent edges of graphs in NAS-Bench-201 as nodes, and nodes of them as edges. Additionally, we add a starting node and an ending node to the cell during training. All nodes which have no predecessors (suceessors) are connected to the starting (ending) node, which we delete after generating the full neural architectures.

E Experimental Setup

E.1 Dataset

1) CIFAR-10 (Krizhevsky et al., 2009): This dataset is a popular benchmark dataset for NAS, which consists of 32×32 colour images from 10 general object classes. The training set consists of 50K images, 5K for each class, and the test set consists of 10K images, 1K for each class. **2) CIFAR-100** (Krizhevsky et al., 2009): This dataset consists of colored images from 100 fine-grained general

object classes. Each class has 500/100 images for training and test, respectively. **3) MNIST** (LeCun & Cortes, 2010): This is a standard image classification dataset which contains 70K 28×28 grey colored images that describe 10 digits. We upsample the images to 32×32 pixels to satisfy the minimum required pixel size of the NAS-Bench 201 due to the residual blocks in the macro skeleton. We use the training/test split from the original dataset, where 60K images are used for training and 10K are used for test. **4) SVHN** (Netzer et al., 2011): This dataset consists of 32×32 color images where each has a digit with a natural scene background. The number of classes is 10 denoting from digit 1 to 10 and the number of training/test images is 73257/26032, respectively. **5) Aircraft** (Maji et al., 2013) This is fine-grained classification benchmark dataset containing 10K images from 30 different aircraft classes. We resize all images into 32×32. **6) Oxford-IIIT Pets** (Parkhi et al., 2012) This dataset is for fine-grained classification which has 37 breeds of pets with roughly 200 instances for each class. There is no split file provided, so we use the 85% of the dataset for training and the other 15% are as a test set. We also resize all images into 32×32. For **CIFAR10** and **CIFAR100**, we used the training, validation, and test splits from the NAS-Bench-201, and use random validation/test splits for **MNIST**, **SVHN**, **Aircraft**, and **Oxford-IIIT Pets** by splitting the test set into two subsets of the same size. The validation set is used to update the searching algorithms as a supervision signal and the test set is used to evaluate the performance of the searched architectures.

E.2 BASELINES

We now briefly describe the baseline models and our MetaD2A model. **1) ResNet** (He et al., 2016) This is a convolutional network which connects the output of previous layer as input to the current layer. It has achieved impressive performance on many challenging image tasks. We use ResNet56 in all experiments. **2) REA** (Real et al., 2019) This is an evolutional-based search method by using aging based tournament selection, showing evolution can work in NAS. **3) RS** (Bergstra & Bengio, 2012) This is based on random search and we randomly samples architectures until the total time of training and evaluation reaches the budget. **4) REINFORCE** (Williams, 1992) This is a RL-based NAS. We reward the model with the validation accuracy after 12 epochs of training. **5) BOHB** (Falkner et al., 2018) This combines the strengths of tree-structured parzen estimator based baysian optimization and hyperband, performing better than standard baysian optimization methods. **5) RSPS** (Li & Talwalkar, 2019) This method is a combination of random search and weight sharing, which trains randomly sampled sub-graphs from weight shared DAG of the search space. The method then selects the best performing sub-graph among the sampled ones as the final neural architecture. **6) SETN** (Dong & Yang, 2019a) SETN is an one-shot NAS method, which selectively samples competitive child candidates by learning to evaluate the quality of the candidates based on the validation loss. **7) GDAS** (Dong & Yang, 2019b) This is a Gumbel-Softmax based differentiable neural architecture sampler, which is trained to minimize the validation loss with the architecture sampled from DAGs. **8) PC-DARTS** (Xu et al., 2020) This is a gradient-based NAS which partially samples channels to apply operations, to improve the efficiency of NAS in terms of memory usage and search time compared to DARTS. We exploit the code at `https://github.com/yuhuixu1993/PC-DARTS`. **9) DrNAS** (Chen et al., 2021) This is a NAS approach that introduces Dirichlet distribution to approximate the architecture distribution, to enhance the generalization performance of differentiable architecture search. We use the code at `https://github.com/xiangning-chen/DrNAS`. We report the results on **CIFAR10** and **CIFAR100** in this paper using the provided code from the authors on the split set of NAS-Bench 201 while their reported results in the paper of the authors are 94.37 and 73.51, respectively on random training/test splits on **CIFAR10** and **CIFAR100**. **10) MetaD2A (Ours)** This is our meta-NAS framework described in section 3, which can stochastically generate task-dependent computational graphs from a given dataset, and use the performance predictor to select the best performing candidates. We follow the same settings of NAS-Bench-201 (Dong & Yang, 2020) for all baselines and use the code at `https://github.com/D-X-Y/AutoDL-Projects` except for 8), 9) and 10).

E.3 IMPLEMENTATION DETAILS

Hyperparameter	Value
The number of inputs of class b	20
Dimension of $\mathbf{v}_c$ $\mathrm{d}_{\mathbf{v}_c}$	56
Dimension of $\mathbf{h}_e$ $\mathrm{d}_{\mathbf{h}_e}$	56
Dimension of $\mathbf{S}$ $\mathrm{d}_{\mathbf{S}}$	56
Dimension of $\mathbf{z}$ $\mathrm{d}_{\mathbf{z}}$	56
Dimension of $\mathbf{h}_{v_i}$ for generator	56
Dimension of $\mathbf{h}_{v_i}$ for predictor	512
The number of operators n_o	5
Learning rate	1e-4
Batch size	32
KL-divergence weighting value λ	5e-3
Training epoch	400

Table 6: Hyperparameter setting of MetaD2A on NAS-Bench-201 Search Space

We use embedding features as inputs of the proposed set encoder instead of raw images, where the embedding features are generated by ResNet18 (He et al., 2016) pretrained with ImageNet-1K (Deng et al., 2009). We adopt the teacher forcing training strategy (Jin et al., 2018), which performs the current decoding process after correcting the decoded graph as the true graph until the previous step. This strategy is only used during meta-training and we progress subsequent generation based on the currently decoded graph part without the true graph information in the meta-test. We use mini-batch gradient descent to train the model with Eq. (1). The values of hyperparameters which we used for both MetaD2A generator and predictor in this paper are described in Table 6. To train searched neural architectures for all datasets, we follow the hyperparameter setting of NAS-Bench-201 (Dong & Yang, 2020), which is used for training searched neural architectures on CIFAR10 and CIFAR100. While we report accuracy after training 50 epoch for MNIST, the accuracy of 200 epoch are reported for all datasets except MNIST.