Rapid Neural Architecture Search by Learning to Generate Graphs from Datasets

Hayeon Lee¹;  Eunyoung Hyung¹*,  Sung Ju Hwang¹,²
KAIST¹, AITRICS², 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 mon-
Conventional NAS Approach

Training NAS Model... Search Cost O(N)

Target Dataset 1 Target Dataset 2 Target Dataset N

Our NAS Approach

Rapid Search

Our Meta-Trained NAS Model

No Retraining on Target

Target Dataset 1 Target Dataset 2 Target Dataset N

Search Cost O(1)

Our Meta-Learning Framework

Source Database

Search Space

Valid Zone

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.

etary budget are limited. How can we then exploit the vast knowledge of neural architectures that have 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.
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, our method 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 scratch for each new unseen task repeatedly and each will take a few GPU hours. The accuracy-predictor-based transference 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 (Elksen 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 (Elksen 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 $N_r$ number of tasks, where each task $\tau = \{D, G, s\}$ consists of a dataset $D$, a neural architecture represented as a Directed Acyclic Graph (DAG) $G$ and an accuracy $s$ obtained from the neural architecture $G$ trained on $D$. 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 $Z$ of datasets and neural architectures from the source database. For each task $\tau$, the generator encodes dataset $D$ as a vector $z$ through the set encoder $q_\phi(z|D)$ parameterized by $\phi$ and then decodes a new graph $\tilde{G}$ from $z$ which are sampled from the prior $p(z)$ by using the graph decoder $p_\theta(\tilde{G}|z)$ parameterized by $\theta$. Then, our goal is that $\tilde{G}$ generated from $D$ to be the true $G$ which is pair of $D$. We meta-learn the generator using set-amortized inference, by maximizing the approximated evidence lower bound (ELBO) as follows:

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

(1)

where

$$\mathcal{L}_{\phi, \theta}(D, G) = \mathbb{E}_{z \sim q_\phi(z|D)} \left[ \log p_\theta(\tilde{G}|z) \right] - \lambda \cdot L_{KL}[q_\phi(z|D)||p(z)]$$

(2)

Each dimension of the prior $p(z)$ factorizes into $\mathcal{N}(0, 1)$. $L_{KL}$ is the KL divergence between two multivariate Gaussian distributions which has a simple closed form (Kingma & Welling, 2014) and $\lambda$ is the scalar weighting value. Using the reparameterization trick on $z$, 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 $f_\omega(s|D, G)$ 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 $f_\omega$ 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 $D$ as an input. We adopt direct acyclic graph encoder (Zhang et al., 2019) for DAG $G$ (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 $f_\omega$ to minimize the MSE loss $\mathcal{L}_{\omega}(s, D, G)$ between the predicted accuracy and the true accuracy $s$ of the model on each task sampled from the source database:

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

(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 \( D \) into a single latent code \( z \), 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 \( D = \{ \mathbf{X}, \mathbf{Y} \} \), where \( \mathbf{X} = \{ \mathbf{X}_c \}_{c=1}^C \) and \( \mathbf{Y} = \{ \mathbf{Y}_c \}_{c=1}^C \) are the set of instances and target labels of \( C \) classes respectively. We randomly sample instances \( \{ \mathbf{x} \mid \mathbf{x} \in \mathbf{B}_c \} \in \mathbb{R}^{b_c \times d_x} \) of class \( c \), where \( \mathbf{x} \) is a \( d_x \) dimensional feature vector, \( \mathbf{B}_c \subset \mathbf{X}_c \) and \( \| \mathbf{B}_c \| = b_c \). We input the sampled instances into the IntraSetPool, the intra-class encoder, to encode class prototype \( \mathbf{v}_c \in \mathbb{R}^{1 \times d_v} \) 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 \( \mathbf{h}_c \in \mathbb{R}^{1 \times d_h} \) as follows:

\[
\mathbf{v}_c = \text{IntraSetPool}(\{ \mathbf{x} \mid \mathbf{x} \in \mathbf{B}_c \}), \quad \mathbf{h}_c = \text{InterSetPool}(\{ \mathbf{v}_c \}_{c=1}^C)
\]  

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 \( \hat{D} \), we can obtain \( n \) set-dependent DAGs \( \{ \hat{G}_1 \}_{i=1}^n \) with the meta-trained generator parameterized by \( \phi^* \) and \( \theta^* \), by feeding \( \hat{D} \) as an input. Through such set-level amortized inference, our method can easily generate neural architecture(s) for the novel dataset. The latent code \( z \in \mathbb{R}^{1 \times d_x} \) can be sampled from a dataset-conditioned Gaussian distribution with diagonal covariance where \( \text{NN}_{\mu}, \text{NN}_{\sigma} \) are single linear layers:

\[
z \sim q_\phi(z | \hat{D}) = \mathcal{N}(\mu, \sigma^2) \quad \text{where} \quad \mu, \sigma = \text{NN}_{\mu}(\mathbf{h}_c), \text{NN}_{\sigma}(\mathbf{h}_c)
\]  

In the meta-test, the predictor \( f_\omega(\hat{s}_i | \hat{D}, \hat{G}_i) \) predicts accuracies \( \{ \hat{s}_i \}_{i=1}^n \) for a given unseen dataset \( \hat{D} \) and each generated architecture of \( \{ \hat{G}_i \}_{i=1}^n \) 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 acyclic 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 zero, 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-set 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 $N_r = 1,310/4,230$ 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 $10^{10}$ 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 $N_r = 3, 018/153$, 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/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.

<table>
<thead>
<tr>
<th>Target Dataset</th>
<th>NAS Method</th>
<th>NAS Training-free on Target</th>
<th>Params (M)</th>
<th>Search Time (GPU Sec)</th>
<th>Speed Up</th>
<th>Search Cost ($)</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>ResNet (He et al., 2016)</td>
<td>✓</td>
<td>0.86</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>93.97 ± 0.00</td>
</tr>
<tr>
<td></td>
<td>REA (Real et al., 2019)</td>
<td>✓</td>
<td>0.02 + T</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>93.92 ± 0.30</td>
</tr>
<tr>
<td></td>
<td>RS (Bergstra &amp; Bengio, 2012)</td>
<td>✓</td>
<td>0.01 + T</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>93.70 ± 0.36</td>
</tr>
<tr>
<td></td>
<td>REINFORCE (Williams, 1992)</td>
<td>✓</td>
<td>0.12 + T</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>93.85 ± 0.37</td>
</tr>
<tr>
<td></td>
<td>BOBB (Falkner et al., 2018)</td>
<td>✓</td>
<td>0.59 + T</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>93.61 ± 0.37</td>
</tr>
<tr>
<td></td>
<td>RPS (Li &amp; Talwalkar, 2019a)</td>
<td>✓</td>
<td>-</td>
<td>107.0</td>
<td>17.7</td>
<td>14.9</td>
<td>54.17 ± 3.57</td>
</tr>
<tr>
<td></td>
<td>SETN (Dong &amp; Yang, 2019a)</td>
<td>✓</td>
<td>-</td>
<td>30200</td>
<td>437</td>
<td>12.55</td>
<td>87.64 ± 0.00</td>
</tr>
<tr>
<td></td>
<td>GDAS (Dong &amp; Yang, 2019b)</td>
<td>✓</td>
<td>-</td>
<td>25077</td>
<td>383</td>
<td>10.17</td>
<td>93.61 ± 0.00</td>
</tr>
<tr>
<td></td>
<td>PC-DARTS (Xu et al., 2020)</td>
<td>✓</td>
<td>1.17</td>
<td>10395</td>
<td>150</td>
<td>4.21</td>
<td>93.66 ± 0.37</td>
</tr>
<tr>
<td></td>
<td>DeNAS (Chen et al., 2021)</td>
<td>✓</td>
<td>1.53</td>
<td>21760</td>
<td>315</td>
<td>8.82</td>
<td>94.36 ± 0.00</td>
</tr>
<tr>
<td></td>
<td>MetaD2A (Ours)</td>
<td>✓</td>
<td>1.11</td>
<td>69</td>
<td>1</td>
<td>0.028</td>
<td>94.37 ± 0.83</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>ResNet (He et al., 2016)</td>
<td>✓</td>
<td>0.86</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>93.69 ± 0.01</td>
</tr>
<tr>
<td></td>
<td>RPS (Li &amp; Talwalkar, 2019)</td>
<td>✓</td>
<td>0.25</td>
<td>32457</td>
<td>3208</td>
<td>9.10</td>
<td>93.63 ± 0.02</td>
</tr>
<tr>
<td></td>
<td>SETN (Dong &amp; Yang, 2019a)</td>
<td>✓</td>
<td>0.56</td>
<td>69656</td>
<td>9950</td>
<td>28.24</td>
<td>93.69 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>GDAS (Dong &amp; Yang, 2019b)</td>
<td>✓</td>
<td>0.82</td>
<td>60186</td>
<td>8598</td>
<td>23.85</td>
<td>95.09 ± 0.24</td>
</tr>
<tr>
<td></td>
<td>PC-DARTS (Xu et al., 2020)</td>
<td>✓</td>
<td>0.62</td>
<td>24857</td>
<td>3551</td>
<td>10.08</td>
<td>95.60 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>DeNAS (Chen et al., 2021)</td>
<td>✓</td>
<td>1.53</td>
<td>44131</td>
<td>6304</td>
<td>17.89</td>
<td>95.99 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>MetaD2A (Ours)</td>
<td>✓</td>
<td>0.61</td>
<td>7</td>
<td>1</td>
<td>0.002</td>
<td>95.71 ± 0.18</td>
</tr>
<tr>
<td>MNIST</td>
<td>ResNet (He et al., 2016)</td>
<td>✓</td>
<td>0.86</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>96.13 ± 0.10</td>
</tr>
<tr>
<td></td>
<td>RPS (Li &amp; Talwalkar, 2019)</td>
<td>✓</td>
<td>0.25</td>
<td>32457</td>
<td>3208</td>
<td>9.10</td>
<td>96.03 ± 0.03</td>
</tr>
<tr>
<td></td>
<td>SETN (Dong &amp; Yang, 2019a)</td>
<td>✓</td>
<td>0.48</td>
<td>85189</td>
<td>12169</td>
<td>34.54</td>
<td>96.02 ± 0.12</td>
</tr>
<tr>
<td></td>
<td>GDAS (Dong &amp; Yang, 2019b)</td>
<td>✓</td>
<td>0.56</td>
<td>69656</td>
<td>9950</td>
<td>28.24</td>
<td>96.04 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>PC-DARTS (Xu et al., 2020)</td>
<td>✓</td>
<td>0.47</td>
<td>31124</td>
<td>4446</td>
<td>12.62</td>
<td>95.40 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>DeNAS (Chen et al., 2021)</td>
<td>✓</td>
<td>1.53</td>
<td>52791</td>
<td>7541</td>
<td>21.40</td>
<td>96.30 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>MetaD2A (Ours)</td>
<td>✓</td>
<td>0.67</td>
<td>7</td>
<td>1</td>
<td>0.004</td>
<td>96.34 ± 0.07</td>
</tr>
<tr>
<td>SVHN</td>
<td>ResNet (He et al., 2016)</td>
<td>✓</td>
<td>0.86</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>93.69 ± 0.10</td>
</tr>
<tr>
<td></td>
<td>RPS (Li &amp; Talwalkar, 2019)</td>
<td>✓</td>
<td>0.22</td>
<td>18697</td>
<td>1869</td>
<td>7.58</td>
<td>42.19 ± 0.88</td>
</tr>
<tr>
<td></td>
<td>SETN (Dong &amp; Yang, 2019a)</td>
<td>✓</td>
<td>0.48</td>
<td>85189</td>
<td>12169</td>
<td>34.54</td>
<td>96.02 ± 0.12</td>
</tr>
<tr>
<td></td>
<td>GDAS (Dong &amp; Yang, 2019b)</td>
<td>✓</td>
<td>0.56</td>
<td>69656</td>
<td>9950</td>
<td>28.24</td>
<td>96.04 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>PC-DARTS (Xu et al., 2020)</td>
<td>✓</td>
<td>0.47</td>
<td>31124</td>
<td>4446</td>
<td>12.62</td>
<td>95.40 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>DeNAS (Chen et al., 2021)</td>
<td>✓</td>
<td>1.53</td>
<td>52791</td>
<td>7541</td>
<td>21.40</td>
<td>96.30 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>MetaD2A (Ours)</td>
<td>✓</td>
<td>0.67</td>
<td>7</td>
<td>1</td>
<td>0.004</td>
<td>96.34 ± 0.07</td>
</tr>
<tr>
<td>Aircraft</td>
<td>ResNet (He et al., 2016)</td>
<td>✓</td>
<td>0.86</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>47.01 ± 1.36</td>
</tr>
<tr>
<td></td>
<td>RPS (Li &amp; Talwalkar, 2019)</td>
<td>✓</td>
<td>0.22</td>
<td>18697</td>
<td>1869</td>
<td>7.58</td>
<td>42.19 ± 0.88</td>
</tr>
<tr>
<td></td>
<td>SETN (Dong &amp; Yang, 2019a)</td>
<td>✓</td>
<td>0.48</td>
<td>85189</td>
<td>12169</td>
<td>34.54</td>
<td>96.02 ± 0.12</td>
</tr>
<tr>
<td></td>
<td>GDAS (Dong &amp; Yang, 2019b)</td>
<td>✓</td>
<td>0.56</td>
<td>69656</td>
<td>9950</td>
<td>28.24</td>
<td>96.04 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>PC-DARTS (Xu et al., 2020)</td>
<td>✓</td>
<td>0.47</td>
<td>31124</td>
<td>4446</td>
<td>12.62</td>
<td>95.40 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>DeNAS (Chen et al., 2021)</td>
<td>✓</td>
<td>1.53</td>
<td>52791</td>
<td>7541</td>
<td>21.40</td>
<td>96.30 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>MetaD2A (Ours)</td>
<td>✓</td>
<td>0.67</td>
<td>7</td>
<td>1</td>
<td>0.004</td>
<td>96.34 ± 0.07</td>
</tr>
<tr>
<td>Oxford-IIIT Pets</td>
<td>ResNet (He et al., 2016)</td>
<td>✓</td>
<td>0.86</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>25.58 ± 0.47</td>
</tr>
<tr>
<td></td>
<td>RPS (Li &amp; Talwalkar, 2019)</td>
<td>✓</td>
<td>0.22</td>
<td>18697</td>
<td>1869</td>
<td>7.58</td>
<td>42.19 ± 0.88</td>
</tr>
<tr>
<td></td>
<td>SETN (Dong &amp; Yang, 2019a)</td>
<td>✓</td>
<td>0.48</td>
<td>85189</td>
<td>12169</td>
<td>34.54</td>
<td>96.02 ± 0.12</td>
</tr>
<tr>
<td></td>
<td>GDAS (Dong &amp; Yang, 2019b)</td>
<td>✓</td>
<td>0.56</td>
<td>69656</td>
<td>9950</td>
<td>28.24</td>
<td>96.04 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>PC-DARTS (Xu et al., 2020)</td>
<td>✓</td>
<td>0.47</td>
<td>31124</td>
<td>4446</td>
<td>12.62</td>
<td>95.40 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>DeNAS (Chen et al., 2021)</td>
<td>✓</td>
<td>1.53</td>
<td>52791</td>
<td>7541</td>
<td>21.40</td>
<td>96.30 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>MetaD2A (Ours)</td>
<td>✓</td>
<td>0.67</td>
<td>7</td>
<td>1</td>
<td>0.004</td>
<td>96.34 ± 0.07</td>
</tr>
</tbody>
</table>

Figure 3: Performance on Unseen Datasets (Meta-Test) We show accuracy over flop of both MetaD2A and a transferable NAS referred to as 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).

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.

Effectiveness of Set-to-Architecture Generator

![Figure 4: T-SNE vis. of Latent Space](image1)

Figure 4: T-SNE vis. of Latent Space

We first visualize cross-modal latent embeddings \( \{ z \} \) of unseen datasets encoded by the meta-
learned generator with T-SNE in Figure 4. Each marker indicates \( z \) of the sampled subsets of each dataset with different seeds. We observe that the generator classifies well embeddings \( \{z\} \) by datasets in the latent space while clusters \( z \) of the subset of the same dataset. Furthermore, we investigate the quality of generated architectures from those embeddings \( \{z\} \). 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.

**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)).

**Table 4: Analysis of Generated Architectures**

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Validity</th>
<th>Uniqueness</th>
<th>Novelty</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAS-Bench-201</td>
<td>1.0000</td>
<td>0.3519</td>
<td>0.6731</td>
</tr>
<tr>
<td>MobileNetV3</td>
<td>0.9831</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Predictor Model</th>
<th>Input Type</th>
<th>Pearson Cor. Coeff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Encoder (GE) Only</td>
<td>✓</td>
<td>0.6439</td>
</tr>
<tr>
<td>DeepSet (Zaheer et al., 2017) + GE</td>
<td>✓</td>
<td>0.7286</td>
</tr>
<tr>
<td>SetTransformer (Lee et al., 2019a) + GE</td>
<td>✓</td>
<td>0.7744</td>
</tr>
<tr>
<td>Statistical Pooling (Lee et al., 2020) + GE</td>
<td>✓</td>
<td>0.7796</td>
</tr>
<tr>
<td><strong>The Proposed Set Encoder + GE (Ours)</strong></td>
<td>✓</td>
<td>0.8085</td>
</tr>
</tbody>
</table>

**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 for 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 conducted by Center for Applied Research in Artificial Intelligence (CARAI) grant funded by DAPA and ADD (UD190031RD).
REFERENCES


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


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.


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


A Details of the Generator

A.1 Graph Decoding

To generate the $i^{th}$ node $v_i$, we compute the operation type $o_{v_i} \in \mathbb{R}^{1 \times n_o}$ over $n_o$ operations based on the current graph state $h_G := h_{v_{i-1}}$ 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 $e_{\{v_i, v_j\}}$, we consider nodes $\{v_j \mid j = i - 1, \ldots, 1\}$ 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 $h_{v_0} = \text{NN}_{\text{init}}(z)$, where $\text{NN}_{\text{init}}$ is an MLP followed by $\text{tanh}$. For $i^{th}$ node $v_i$ according to topological order, we compute the probability of each operation type $o_{v_i} \in \mathbb{R}^{1 \times n_o}$ over $n_o$ operations, given the current graph state as the last hidden node $h_G := h_{v_i}$. That is, $o_{v_i} = \text{NN}_{\text{node}}(h_G)$, where $\text{NN}_{\text{node}}$ is an MLP followed by $\text{softmax}$. 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 $h_{v_i}^{(t)}$ at time step $t$ as follows:

$$h_{v_i}^{(t+1)} = \text{UPDATE}(i, m_{v_i}^{(t)})$$

where $m_{v_i}^{(t)} = \sum_{u \in V_{v_i}} \text{AGGREGATE}(h_u^{(t)})$ \hspace{1cm} (6)

The function UPDATE is a gated recurrent unit (GRU) (Cho et al., 2014), $i$ is the order of $v_i$, and $m_{v_i}^{(t)}$ is the incoming message to $v_i$. The function AGGREGATE consists of mapping and gating functions with MLPs, where $V_{v_i}^{(n)}$ is a set of predecessors with incoming edges to $v_i$. For all previously processed nodes $\{v_j \mid j = i - 1, \ldots, 1\}$, we decide whether to link an edge from $v_j$ to $v_i$ by sampling the edge based on edge connection probability $e_{\{v_j, v_i\}} = \text{NN}_{\text{edge}}(h_j, h_i)$, where $\text{NN}_{\text{edge}}$ is an MLP followed by $\text{sigmoid}$. We update $h_{v_i}$ 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 $\mathbb{E}_{z \sim q_{\theta}(z \mid G)} \left[ \log p_{\theta}(G \mid z) \right]$ of (2) can be rewritten with negative cross-entropy loss $-L_{CE}^T$ for nodes and binary cross-entropy loss $-L_{BCE}^T$ for edges, and we slightly modify it using the generated set-dependent graph $\hat{G}$ and the ground truth graph $G$ as the input as follows:

$$- \sum_{i \in V} \left\{ L_{CE}^T(\hat{o}_i, o_i) + \sum_{j \in V} L_{BCE}^T(\hat{e}_{\{j,i\}}, e_{\{j,i\}}) \right\}$$ \hspace{1cm} (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 $\phi, \theta$, which are shared across all tasks.

B Graph Encoding of the Set-dependent Predictor

For a given graph candidate $\hat{G}$, we sequentially perform message passing for nodes from the predecessors following the topological order of the DAG $\hat{G}$. We iteratively update hidden states $h_{v_i}^{(t)}$ using the Eq. (8) by feeding in its predecessors’ hidden states $\{u \in V_{v_i}^{(n)}\}$.

$$h_{v_i}^{(t+1)} = \text{UPDATE}(y_{v_i}, m_{v_i}^{(t)})$$

where $m_{v_i}^{(t)} = \sum_{u \in V_{v_i}^{(n)}} \text{AGGREGATE}(h_u^{(t)})$ \hspace{1cm} (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 $h_f$. 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 $h_b$, which is the last hidden states of the starting node. We concatenate the outputs $h_f$ of the forward graph encoder and $h_b$ 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:

$$SAB(X) = LN(H + MLP(H))$$

where $H = LN(X + MH(X, X, X))$ (9)

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

$$PMA(X) = LN(H + MLP(H))$$

where $H = LN(X + MH(S, MLP(X), MLP(X)))$ (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 $\pi$ on a set $X$ (Zaheer et al., 2017):

$$E(\{x | x \in \pi X\}) = \pi E(\{x | x \in X\})$$

(11)

Since all of the components in SAB and PMA are row-wise computation functions, SAB and PMA is permutation equivariant 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 (successors) 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 $32 \times 32$ 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

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of inputs of class b</td>
<td>20</td>
</tr>
<tr>
<td>Dimension of $v_c$</td>
<td>56</td>
</tr>
<tr>
<td>Dimension of $h_v$</td>
<td>56</td>
</tr>
<tr>
<td>Dimension of $S$</td>
<td>56</td>
</tr>
<tr>
<td>Dimension of $z$</td>
<td>56</td>
</tr>
<tr>
<td>Dimension of $h_v$ for generator</td>
<td>56</td>
</tr>
<tr>
<td>Dimension of $h_v$ for predictor</td>
<td>512</td>
</tr>
<tr>
<td>The number of operators $n_o$</td>
<td>5</td>
</tr>
<tr>
<td>Learning rate</td>
<td>1e-4</td>
</tr>
<tr>
<td>Batch size</td>
<td>32</td>
</tr>
<tr>
<td>KL-divergence weighting value $\lambda$</td>
<td>5e-3</td>
</tr>
<tr>
<td>Training epoch</td>
<td>400</td>
</tr>
</tbody>
</table>

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.