RAPID NEURAL ARCHITECTURE SEARCH BY LEARNING TO GENERATE GRAPHS FROM DATASETS

Anonymous authors
Paper under double-blind review

ABSTRACT

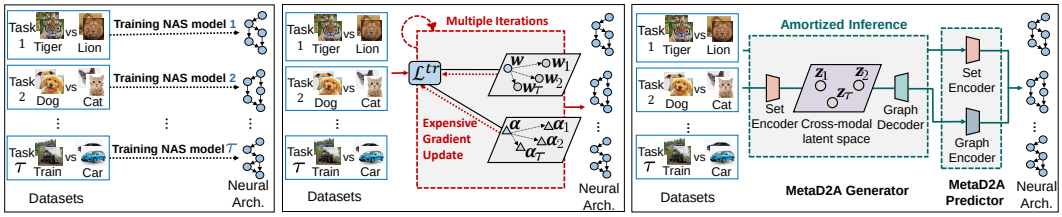
Despite the success of recent Neural Architecture Search (NAS) methods on various tasks which have shown to output networks that largely outperform human-designed networks, conventional NAS methods have mostly tackled the optimization of searching for the network architecture for a single task (dataset), 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 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 from those sampled from MetaD2A. 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 benchmark datasets including CIFAR-10 and CIFAR-100, with the search time of less than 30 GPU seconds on CIFAR-10. 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.

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., 2018a; Luo et al., 2018; Pham et al., 2018; Liu et al., 2019; Xu et al., 2020; Chen et al., 2020).

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 (a)). 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 been already trained on a large number of datasets, to better generalize over an unseen task?

To this end, 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



(a) Task-specific NAS (b) Meta-NAS (c) MetaD2A (Ours)

Figure 1: **Comparison of our method to conventional NAS approaches (a)** Task-specific NAS methods require enormous search time as the search needs to be done from scratch for each task. **(b)** Meta-NAS methods can adapt to new tasks by leveraging meta-knowledge, but they do not scale well to large datasets and networks due to inefficiency in the meta-learning framework. **(c)** Our method can rapidly generate set-specific neural architecture candidates for a given dataset with amortized meta-learning of the cross-modal latent space which maps the dataset to architecture, with the set encoder and the graph decoder. Then, our performance predictor selects the best architecture among the generated architectures based on predicted performance.

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 accuracies predicted by a performance predictor, which is also trained with amortized meta-learning (See Figure 1 (c)).

Several Meta-NAS methods (Lian et al., 2019; Elsken et al., 2020; Shaw et al., 2019) have studied the transferability of NAS to unseen datasets based on gradient-based meta-learning framework. However, their expensive multiple unrolling gradient steps for one meta-update of each task require large computational cost, which limits the applications of those Meta-NAS methods to small scale tasks such as few-shot classification (See Figure 1 (b)). On the other hand, the proposed MetaD2A can output a set-dependent neural architecture for a large dataset, through an amortized inference which encodes the dataset as a latent embedding and decodes it as a directed acyclic graph.

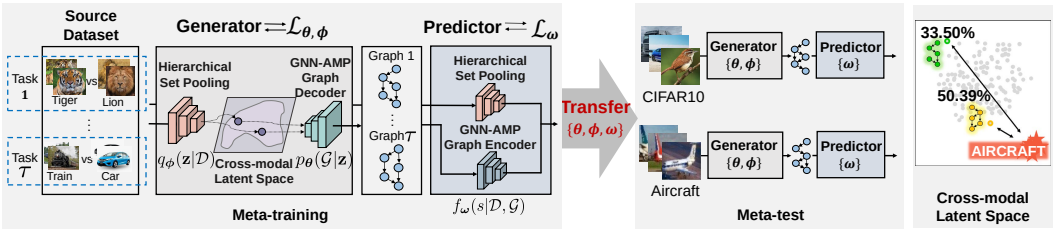
We meta-learn the proposed MetaD2A on MetaImageNet, which is compiled to contain subsets of ImageNet-1K, and neural architectures from the NAS-Bench201 search space. Then we validate it on the architecture search problems for unseen subsets of ImageNet-1K. On this experiment, our model obtains a neural architecture largely outperforming all baselines for a given task within 27.03 GPU seconds on average. Further, our model meta-learned on MetaImageNet can generalize to search for neural architectures on multiple unseen datasets such as MNIST, SVHN, CIFAR-10, CIFAR-100, Aircraft, and Oxford-IIIT Pets. The results show that our model achieves state-of-the-art performance on all datasets. Moreover, our model can search for a neural architecture within 22.47 GPU seconds on average, on CIFAR10.

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 of it 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 22.47 GPU seconds on CIFAR10.

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 for optimal neural architectures. 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), and gradient-based methods (Liu et al., 2019; Cai et al., 2019; Luo et al., 2018; Dong & Yang, 2019b; Chen et al., 2020; Xu et al., 2020; Fang et al., 2020).



(a) (b)

Figure 2: **(a) Overview of MetaD2A** MetaD2A generator with $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ meta-learns the set-dependent DAGs representations on the training tasks, which are the subset of ImageNet-1K. MetaD2A predictor with $\boldsymbol{\omega}$ meta-learns to predict performance across the multiple tasks. We transfer MetaD2A to new target datasets to search for the set-specific architecture within a minute. **(b)** T-SNE visualization of a learned cross-modal latent space.

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. Besides NAO, FNA (Fang et al., 2020) focuses on reducing the adaptation time by using parameter remapping from the pre-trained network, while ours reduces the search time by generating graphs via latent space. 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., 2018a; Pham et al., 2018; Liu et al., 2019; Chen et al., 2020). 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 train a model from the scratch for each new unseen task repeatably and each will take a few GPU hours.

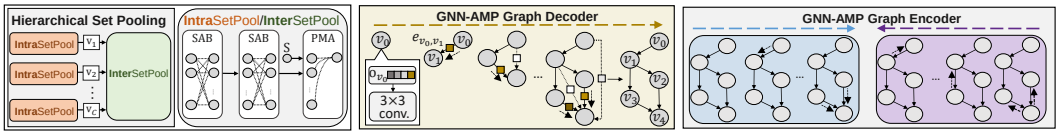
Meta-learning Meta-learning (learning to learn) aims to train a model to generalize over a distribution of tasks, such that it can rapidly adapt to a new task (Vinyals et al., 2016; Snell et al., 2017; Finn et al., 2017; Nichol et al., 2018; Lee et al., 2019b; Hou et al., 2019). Recently, LEO (Rusu et al., 2019) proposed a scalable meta-learning framework which learns the latent generative representations of model parameters for a given data in 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 as 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 its 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 on an unseen input dataset to select the best neural architecture for the dataset. Figure 2 illustrates the overview of our framework.

To formally define the problem, let us assume that we have a database of N_τ number of tasks, where we randomly obtain tasks τ from task distribution $p(\tau)$ to train for the model. Each task $\tau = \{\mathcal{D}, \mathcal{G}, s\}$ consists of a 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, a neural architecture represented as a Directed Acyclic Graph (DAG) $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, and an accuracy $\boldsymbol{s}$ obtained from the neural architecture



(a) Hierarchical Set Pooling (b) GNN-AMP Graph Decoder (c) GNN-AMP Graph Encoder

Figure 3: **(a)** Hierarchical Set Pooling is the attention-based module to extract a set latent vector for DAG generation considering class label information from $\mathcal{D}$. **(b)** GNN-AMP Graph Decoder generates the set-dependent DAG from the latent vector with asynchronous message passing. **(c)** GNN-AMP Graph Encoder for the performance predictor summarizes the input DAG into a single vector with bi-directional encoding scheme.

$\mathcal{G}$ trained on $\mathcal{D}$. Each node $v \in \mathcal{V}$ of $\mathcal{G}$ then represents a computational operator (e.g. 3-by-3 convolution, mean-pooling, ...) and each edge $(u, v) \in \mathcal{E}$ represents a computation flow from an output of the operator u to input of the operator v. The whole framework consists of a set encoder, a graph decoder, and a meta-performance predictor parameterized by $\boldsymbol{\phi}$, $\boldsymbol{\theta}$, and $\boldsymbol{\omega}$, respectively. The dataset $\mathcal{D}$ is embedded into a latent space $\mathcal{Z}$ as a vector $\mathbf{z}$ through the set encoder $q_{\boldsymbol{\phi}}$ and then the graph decoder $p_{\boldsymbol{\theta}}$ estimates the distribution and generates a new graph $\tilde{\mathcal{G}}$ from $\mathbf{z}$ which are sampled from prior the $p(\mathbf{z})$. We learn both components $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})$ and $p_{\boldsymbol{\theta}}(\mathcal{G}|\mathbf{z})$, generalizing over the task distribution $p(\tau)$. After training, for an unseen dataset $\hat{\mathcal{D}}$, our model with $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ can easily generate DAGs $\{\hat{\mathcal{G}}_i\}_{i=1}^{n_s}$ as candidates, where n_s is the number of sampled architectures. This is done by decoding the latent embeddings $\{\hat{\mathbf{z}}_i|\hat{\mathbf{z}}_i \in \mathcal{Z}\}_{i=1}^{n_s}$ sampled from the prior $p(\mathbf{z})$ consisting of $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$ generated from the set encoder for given $\hat{\mathcal{D}}$. The predictor $f_{\boldsymbol{\omega}}(s|\mathcal{D}, \mathcal{G})$ is also learned from the task distribution $p(\tau)$ in the meta-training stage and then selects the best performing neural architecture for $\hat{\mathcal{D}}$ from $\{\hat{\mathcal{G}}_i\}_{i=1}^{n_s}$ based on their predicted accuracies $\{\hat{s}_i\}_{i=1}^{n_s}$ in the meta-test.

3.1 Learning to Generate Graphs from Datasets

Dataset Encoding 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). While simple functions such as mean/sum-pooling can fulfill those requirements, they cannot model high-order interactions between the set elements to accurately generate complex DAGs such as neural architectures. Further, those non-hierarchical poolings consider all elements equally, and thus cannot accurately model individual classes in the given dataset, which may lead to poor performance (Lee et al., 2020).

Therefore, we introduce a novel **Hierarchical Set Pooling** for dataset encoding which stacks two permutation-invariant encoders with learnable parameters, that can extract information related to generating set-conditioned DAGs. 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. Specifically, for a given dataset $\mathcal{D}$, we construct the subset for class c as $\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c\} \in \mathbb{R}^{\mathrm{b} \times \mathrm{d_x}}$, where $\boldsymbol{B}_c \subset \boldsymbol{X}_c$ and $||\boldsymbol{B}_c|| = \mathrm{b}$ by random sampling. We input the sub-dataset into the IntraSetPool, the intra-class encoder, to encode class prototype $\mathbf{v}_c \in \mathbb{R}^{1 \times \mathrm{d_{v_c}}}$ for each class $c = 1, ..., C$: $\mathbf{v}_c = \text{IntraSetPool}\big(\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c\}\big)$. 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 $\mathbf{h}_e \in \mathbb{R}^{1 \times \mathrm{d_{h_e}}}$: $\mathbf{h}_e = \text{InterSetPool}\big(\{\mathbf{v}_c\}_{c=1}^{C}\big)$.

Both the set encoders are comprised of two building blocks: 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. Please see Section A.1 of the Appendix for more details. We set $k = 1$ to produce a single vector, and stack SAB twice followed by PMA for both set encoders as follows (See Figure 3 (a)):

$$\mathbf{v}_c = \text{IntraSetPool}\big(\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c\}\big) = \text{PMA}\big(\text{SAB}\big(\text{SAB}\big(\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c\}\big)\big)\big) \tag{1}$$

$$\mathbf{h}_e = \text{InterSetPool}\big(\{\mathbf{v}_c\}_{c=1}^{C}\big) = \text{PMA}\big(\text{SAB}\big(\text{SAB}\big(\{\mathbf{v}_c\}_{c=1}^{C}\big)\big)\big) \tag{2}$$

Then 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:

$$\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2), \text{where } \boldsymbol{\mu}, \boldsymbol{\sigma} = \text{NN}_{\boldsymbol{\mu}}(\mathbf{h}_e), \text{NN}_{\boldsymbol{\sigma}}(\mathbf{h}_e) \tag{3}$$

Graph Decoding To generate a DAG $\mathcal{G}$ describing a neural architecture from the latent code $\mathbf{z}$, we use a Graph Neural Network(GNN)-based decoder $p_{\boldsymbol{\theta}}(\mathcal{G}|\mathbf{z})$. While GNN-based decoders have

been actively used to generate graph-structured data (Liu et al., 2018b; Liao et al., 2019; Jin et al., 2020), they are suboptimal for our goal since they generally target undirected graphs and therefore ignore the *topological order* of DAGs. Thus, we use a **GNN with asynchronous message passing (GNN-AMP)** which allows message passing to happen only along the *topological order* of the DAGs (Zhang et al., 2019). The graph decoder starts from an initial hidden state $\mathbf{h}_{v_0} = \text{NN}_{\text{init}}(\mathbf{z})$, where NN_{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 $\text{o}_{v_i} \in \mathbb{R}^{1\times n_o}$ over n_o operations, given the current graph state as the last hidden node $\mathbf{h}_\mathcal{G} := \mathbf{h}_{v_i}$. That is, $\text{o}_{v_i} = \text{NN}_{\text{node}}(\mathbf{h}_\mathcal{G})$, where NN_{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:

$$\mathbf{h}_{v_i}^{(t+1)} = \text{UPDATE}(i, \boldsymbol{m}_{v_i}^{(t)}), \text{ where } \boldsymbol{m}_{v_i}^{(t)} = \sum_{u\in\mathcal{V}_{v_i}^{in}} \text{AGGREGATE}(\mathbf{h}_u^{(t)}) \tag{4}$$

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 $e_{\{v_j,v_i\}} = \text{NN}_{\text{edge}}(\mathbf{h}_j, \mathbf{h}_i)$, where NN_{edge} is a MLP followed by $sigmoid$. We update $\mathbf{h}_{v_i}$ by Eq. (4) 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 $e_{\{v_j,v_i\}} > 0.5$. Fig 3(b) illustrates the decoder. For more details, please see Section A.2.

Meta-training Strategy for NAS We meta-learn the model using set-amortized inference, by maximizing the approximated evidence lower bound (ELBO) for each task τ as follows:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})}\big[\log p_{\boldsymbol{\theta}}(\mathcal{G}|\mathbf{z})\big] - \lambda \cdot L_{KL}^{\tau}\big[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})||p(\mathbf{z})\big] \tag{5}$$

where each dimension of prior $p(\mathbf{z})$ factorizes into $\mathcal{N}(0,1)$. L_{KL}^{τ} 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 can optimize the above objective by stochastic gradient variational Bayes (Kingma & Welling, 2014). The expectation of the log-likelihood of (5) 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:

$$-L_{\boldsymbol{\phi},\boldsymbol{\theta}}^{\tau}(\mathcal{D},\mathcal{G}) = \sum_{v_i\in\mathcal{V}} \Big\{ -L_{CE}^{\tau}(\tilde{o}_{v_i}, o_{v_i}) + \sum_{v_j\in\mathcal{V}_i} -L_{BCE}^{\tau}(\tilde{e}_{\{v_j,v_i\}}, e_{\{v_j,v_i\}}) \Big\} \tag{6}$$

We substitute the log-likelihood term of the equation (5) such as equation (6) and take negative to the equation (5) to convert it to the resultant objective $\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}^{\tau}(\mathcal{D},\mathcal{G})$ as follows:

$$\min_{\boldsymbol{\phi},\boldsymbol{\theta}} \sum_{\tau\sim p(\tau)} \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}^{\tau}(\mathcal{D},\mathcal{G}) = \min_{\boldsymbol{\phi},\boldsymbol{\theta}} \sum_{\tau\sim p(\tau)} \Big\{ L_{\boldsymbol{\phi},\boldsymbol{\theta}}^{\tau}(\mathcal{D},\mathcal{G}) + \lambda \cdot L_{KL}^{\tau}\big[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})||p(\mathbf{z})\big] \Big\} \tag{7}$$

We learn our MetaD2A generator by minimizing the final objective to learn $\boldsymbol{\phi}, \boldsymbol{\theta}$, which are shared across all tasks. In the meta-test stage, for a unseen dataset $\hat{\mathcal{D}}$, we can obtain set-dependent DAGs $\{\hat{\mathcal{G}}_i\}_{i=1}^{n_s}$, with the meta-trained generator parameterized by $\boldsymbol{\phi}^*, \boldsymbol{\theta}^*$, by feeding $\hat{\mathcal{D}}$ as the input. Thus our method can generate neural architecture(s) for the novel dataset via set-level amortized inference.

3.2 Meta-Performance Predictor

We design a meta-performance predictor f_ω to select the best architecture from the architectures generated by MetaD2A generator, based on their predicted accuaracies on the target task. Since our predictor should accurately predict the model performance for any datasets unlike the predictors used in task-specific NAS, it also meta-learned. The meta-performance predictor consists of a dataset encoder and a graph encoder, followed by two linear layers with $relu$. For dataset encoding, we use the **Hierarchical Set Pooling** of Section 3.1 which takes $\mathcal{D}$ as an input, using only the mean $\boldsymbol{\mu}$ from Eq. (3) as an output. We perform a bi-directional encoding of the given graph following **GNN-AMP** (Zhang et al., 2019), where given $\mathcal{G}$, we sequentially perform message passing for nodes from

the predecessors following the *topological order* of the DAG $\mathcal{G}$ and then reverse the node orders and perform an additional encoding process (See Figure 3 (c), and Section A.3 of the appendix for more details). We concatenate the outputs of both the bi-directional 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 between the predicted accuracy and the true accuracy s_i of the model on each task:

$$\min_{\boldsymbol{\phi},\boldsymbol{\theta}} \sum_{\tau \sim p(\tau)} \mathcal{L}^{\tau}_{\boldsymbol{\omega}}(s, \mathcal{D}, \mathcal{G}) = \min_{\boldsymbol{\phi},\boldsymbol{\theta}} \sum_{\tau \sim p(\tau)} (s - f_{\boldsymbol{\omega}}(\mathcal{D}, \mathcal{G}))^2 \tag{8}$$

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

To our knowledge, the performance predictors proposed in the existing NAS works need re-training for the new dataset since they have no module to handle multiple datasets but only encoders for graphs. On the other hand, our proposed meta-performance predictor consists of a **dataset encoder** and **graph encoder** to address such a problem with one training. Moreover, the proposed meta-performance predictor learns meta-knowledge accumulated from multiple tasks of MetaImageNet. Thus, the predictor can predict performance for the new unseen dataset. As the number of target datasets increases, the time complexity of other predictors linearly increases while ours needs a constant time complexity of O(1). In addition, none of the existing works in both NAS or meta-learning can handle the scale of meta-learning as our MetaD2A does.

3.3 Comparision with Conventional Meta-NAS

Existing meta-NAS approaches (Elsken et al., 2020; Lian et al., 2019; Shaw et al., 2019) mainly target few-shot classification and cannot scale to such a large task with a large number of samples, since they rely on gradient-based meta-learning with rollout gradient steps. For few-shot learning, taking a few gradient steps is more than enough since there are only a few samples from which we can compute the gradients, but for large-scale learning, taking few gradient steps with minibatch sampling will only consider a very small portion of the samples in the task, which will be largely insufficient for the model to converge. Thus the model needs to take a large number of gradient steps with a large number of samples, however, this is extremely costly and almost infeasible when computing second-order derivatives as in MAML. On the contrary, MetaD2A can handle the large-way many-shot problem tackled by general classification tasks, since it does not require any rollout gradient steps when inferring the architecture. For conventional few-shot settings (5-way 5-shot), the number of samples per task is 25. Contrarily, the number of samples of CIFAR-100 used in MetaD2A is $100 \times 20 = 2,000$ which is 80 times larger. Further, the number of instances(e.g. 20) can be set to a much larger number. Overcoming such a limitation of conventional meta-NAS methods, and meta-learning in general, is one of the main contributions of this work, which is highly novel, in both the perspective of NAS and meta-learning. Further, to compare MetaD2A with meta-NAS methods, we perform experiments on few-shot tasks in the Section D of the Appendix.

4 Experiment

We conduct extensive experiments to demonstrate the effectiveness and efficiency of MetaD2A.

Search Space We explore the same search space of NAS-Bench-201 (Dong & Yang, 2020), cell-based NAS, for all experiments. (See the Section B of the Appendix for more details).

Dataset We validate our method on multiple datasets as follows. **1) MetaImageNet**: To meta-learn our model on multiple tasks, we compiled a new dataset consisting of multiple subsets of ImageNet-1K (Deng et al., 2009) and their neural architectures. We downsample the image size to 32×32 and split classes into 600/400 for meta-training/meta-test, respectively. For each task of meta-training/meta-test, we make a dataset by sampling 20 classes randomly from meta-training/meta-test with an average of 26K images, we search set-specific architecture on the sampled dataset using GDAS (Dong & Yang, 2019b) or random search, and we obtain its accuracy by training the searched architecture on the dataset. We collect N_τ =1,679/3,752 meta-training tasks for the MetaD2A generator/predictor and 20/400 meta-test tasks for them, respectively. We apply our model trained from **MetaImageNet** to multiple benchmark datasets such as **2) MNIST** (LeCun & Cortes, 2010), **3) SVHN** (Netzer et al., 2011), **4) CIFAR-10** (Krizhevsky et al., 2009), **5) CIFAR-100** (Krizhevsky et al., 2009), **6) Aircraft** (Maji et al., 2013), and **7) Oxford-IIIT Pets** (Parkhi et al., 2012). (For detailed descriptions of the baselines, datasets and experimental setup, see Appendix, Section C)

Table 1: **Performance of our model and baselines on unseen MetaImageNet tasks.** We validate all models on 20 unseen tasks from MetaImageNet. We report the mean accuracies and 95% confidence intervals.

Method	Transfer	Search Time (sec)	Search Expense ($)	Params (M)	MetaImageNet Top-1 Accuracy	MetaImageNet Top-5 Accuracy
ResNet (He et al., 2016)	N	N/A	N/A	0.86	$67.12_{\pm 1.23}$	$91.03_{\pm 0.68}$
ENAS (Pham et al., 2018)	N	247479.12	100.37	0.07	$28.05_{\pm 1.25}$	$63.90_{\pm 1.54}$
RSPS (Li & Talwalkar, 2019)	N	11009.18	4.46	0.28	$60.66_{\pm 1.56}$	$89.37_{\pm 0.90}$
DARTS (Liu et al., 2019)	N	9714.14	3.94	0.10	$43.02_{\pm 3.74}$	$76.70_{\pm 2.72}$
SETN (Dong & Yang, 2019a)	N	30738.71	12.47	0.36	$60.87_{\pm 1.54}$	$89.03_{\pm 0.78}$
GDAS (Dong & Yang, 2019b)	N	25272.03	10.25	0.67	$64.59_{\pm 1.36}$	$89.89_{\pm 0.76}$
PC-DARTS (Xu et al., 2020)	N	10721.59	8.39	0.44	$57.62_{\pm 4.66}$	$83.60_{\pm 4.97}$
DrNAS (Chen et al., 2020)	N	10949.95	4.44	1.00	$67.48_{\pm 1.05}$	$91.36_{\pm 0.63}$
MetaD2A (Ours)	Y	27.03 / 26964.18[†]	0.01/10.94	0.72	**$69.36_{\pm 2.16}$**	**$91.91_{\pm 1.21}$**

Table 2: **Performance on multiple datasets** We transfer MetaD2A to target datasets after training on MetaImageNet, while other baselines are trained to search neural architectures for each dataset from scratch. We report accuracies[‡] with 95% confidence intervals on multiple datasets and parameters on CIFAR10.

Method	Params (M)	Target Dataset MNIST	SVHN	CIFAR-10	CIFAR-100	Aircraft	Oxford-IIIT Pets
Optimal	-	-	-	94.37	73.51	-	-
ResNet (He et al., 2016)	0.86	$99.67_{\pm 0.01}$	$96.13_{\pm 0.19}$	$93.97_{\pm 0.00}$	$70.86_{\pm 0.00}$	$47.01_{\pm 1.16}$	$25.58_{\pm 3.43}$
REA (Real et al., 2019)	-	-	-	$93.92_{\pm 0.30}$	$71.84_{\pm 0.99}$	-	-
RS (Bergstra & Bengio, 2012)	-	-	-	$93.70_{\pm 0.36}$	$71.04_{\pm 1.07}$	-	-
REINFORCE (Williams, 1992)	-	-	-	$93.85_{\pm 0.37}$	$71.71_{\pm 1.09}$	-	-
BOHB (Falkner et al., 2018)	-	-	-	$93.61_{\pm 0.52}$	$70.85_{\pm 1.28}$	-	-
DVAE (Zhang et al., 2019)	0.86	-	-	$85.53_{\pm 1.31}$	$51.10_{\pm 3.99}$	-	-
RSPS (Li & Talwalkar, 2019)	-	$99.63_{\pm 0.02}$	$96.17_{\pm 0.12}$	$84.07_{\pm 3.61}$	$52.31_{\pm 5.77}$	$42.19_{\pm 3.88}$	$22.91_{\pm 1.65}$
DARTS (Liu et al., 2019)	-	$98.82_{\pm 0.69}$	$65.71_{\pm 25.88}$	$54.30_{\pm 0.00}$	$15.61_{\pm 0.00}$	$22.50_{\pm 0.04}$	$18.14_{\pm 5.00}$
SETN (Dong & Yang, 2019a)	-	$99.69_{\pm 0.04}$	$96.02_{\pm 0.12}$	$87.64_{\pm 0.00}$	$59.05_{\pm 0.24}$	$44.84_{\pm 3.96}$	$25.17_{\pm 1.68}$
GDAS (Dong & Yang, 2019b)	-	$99.64_{\pm 0.04}$	$95.57_{\pm 0.57}$	$93.61_{\pm 0.09}$	$70.70_{\pm 0.30}$	$53.52_{\pm 0.48}$	$24.02_{\pm 2.75}$
PC-DARTS (Xu et al., 2020)	1.18	$99.66_{\pm 0.04}$	$95.40_{\pm 0.67}$	$93.19_{\pm 0.69}$	$68.89_{\pm 0.23}$	$26.33_{\pm 3.40}$	$19.25_{\pm 8.52}$
DrNAS (Chen et al., 2020)	1.53	$99.60_{\pm 0.04}$	$96.28_{\pm 0.01}$	$93.76_{\pm 0.00}$	$71.50_{\pm 0.75}$	$40.45_{\pm 9.06}$	$26.73_{\pm 2.61}$
FNA (Fang et al., 2020)	0.07	$99.51_{\pm 0.16}$	$94.19_{\pm 0.12}$	$70.41_{\pm 0.77}$	-	$26.43_{\pm 3.37}$	$13.57_{\pm 0.41}$
MetaD2A (Ours)	1.05	**$99.70_{\pm 0.18}$**	**$96.56_{\pm 0.13}$**	**$94.37_{\pm 0.01}$**	**$73.51_{\pm 0.00}$**	**$60.31_{\pm 0.74}$**	**$38.74_{\pm 0.08}$**

4.1 SEARCH RESULTS ON METAIMAGENET

In Table 1, we report the performance of baselines and our model trained on the training set of MetaImageNet, on unseen tasks of MetaImageNet. Note that for the unseen MetaImageNet tasks, **all** baseline models * require to train the NAS model from scratch on each individual task . On the contrary, our model can generate set-dependent architectures with the amortized inference through the architecture generator. In the inference process, we sample 1000 architecture candidates from the MetaD2A generator and select and train one architecture with the highest predicted accuracy computed by the meta-performance predictor. We report the average accuracy of over 20 unseen tasks with 3 different random seeds for all models. For our model, we report both the search time (27.03s) averaged on the 20 tasks, and meta-training time (26964.18s)[†]. We calculate the search expense by with the hourly cost of 1.46$/h, of Tesla P100 on Google Cloud. All models are run on an Nvidia 2080ti GPU. The result shows that the proposed MetaD2A outperforms all baselines in both the accuracy and search time on unseen tasks, with fast search time (27.03s). We further report the accuracy over the search time in Figure 4 (c). Once meta-training is done, our model can generate architectures on any new tasks without training the NAS framework again, which can be done at extremely low cost ($0.01 for each dataset), which makes it highly practical.

4.2 SEARCH RESULTS ON MULTIPLE DATASETS

Table 2 shows that our model meta-trained on the MetaImageNet dataset can generalize to multiple datasets such as MNIST, SVHN, CIFAR-10, CIFAR-100, Aircraft, and Oxford-IIIT Pets. Since MetaD2A can output set-adaptive architectures on the target datasets, there is **no** training cost for the unseen tasks. MetaD2A infers fast, as Table 3 shows, such as 22.47(s) for **CIFAR10** and it has relatively comparable parameters 1.05M for **CIFAR10**. On **CIFAR10** and **CIFAR100**, we report the mean accuracies over 10 runs of the search process by retrieving accuracies of searched architectures from NAS-Bench-201. Specifically, following SETN (Dong & Yang, 2019a), we retrieve the accuracies of N architecture candidates from the NAS-bench-201 and report the highest final accu-

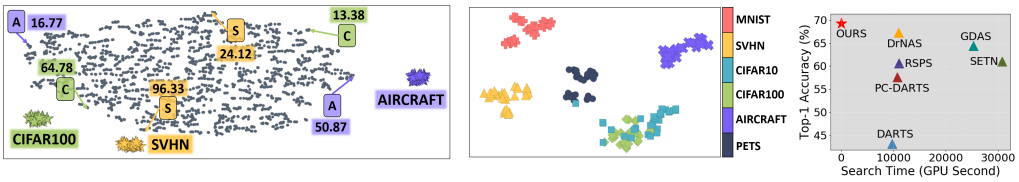
*We do not compare against existing Meta-NAS methods since they cannot scale to non-fewshot settings.

[†]Note that the meta-training is done **only once** throughout all experiments in the paper.

[‡]For some baseline models, we brought the results for CIFAR-10 and CIFAR-100 from NAS-Bench-201 (Dong & Yang, 2020) which performs the experiments with a single GeForce GTX 1080 Ti GPU.

Table 3: **Architecture Search Time on multiple datasets** We report architecture search time(s) of each model on various datasets. After once meta-trained on MetaImageNet, we transfer MetaD2A to target datasets, while other baselines are trained to search neural architectures for each dataset from scratch.

Method	Target Dataset					
	MNIST	SVHN	CIFAR-10	CIFAR-100	Aircraft	Oxford-IIIT Pets
RSPS (Li & Talwalkar, 2019)	22457.20	27962.50	10200.10	18841.70	18697.40	3360.20
DARTS (Liu et al., 2019)	22398.40	26900.60	8007.13	18203.70	3624.70	2469.40
SETN (Dong & Yang, 2019a)	69656.20	85189.60	30220.70	58808.70	18564.40	8625.20
GDAS (Dong & Yang, 2019b)	60186.50	71595.00	25077.20	51580.00	18508.20	6965.40
PC-DARTS (Xu et al., 2020)	24857.80	31124.50	10395.00	19951.50	3524.30	2844.60
DrNAS (Chen et al., 2020)	44131.00	52791.00	21760.00	34529.00	32422.00	6019.50
FNA (Fang et al., 2020)	22830.40	27087.90	9462.80	-	3972.80	2619.50
MetaD2A (Ours)	**32.95**	**30.20**	**22.47**	**68.58**	**34.51**	**38.79**



(a) (b) (c)

Figure 4: **Qualitative Examples (a)** T-SNE vis. of **z** of datasets (stars) and graphs (grey circles). **(b)** T-SNE vis. of **z** of several datasets. **(c)** Mean Accuracy over Search Time on unseen MetaImageNet tasks. The actual architectures and more examples of (a) and (b) are in the Section D of the Appendix.

racy for each run. While $N = 1000$ in SETN, MetaD2A searches for an optimal architecture using a smaller number of samples ($N = 40$), reliably returning 94.37 and 73.51 over 10 runs, in very small confidence interval. $N = 10$ is sufficient to obtain the model that outperforms the baselines as shown in Figure 5. On **MNIST**, **SVHN**, **Aircraft**, and **Oxford-IIIT Pets**, we search one architecture by MetaD2A model and report the accuracy averaging over 3 runs with different seeds. 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. These experiments show the transferability of MetaD2A, as obtaining higher accuracy against all baselines on multiple 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**.

Qualitative analysis We further perform several qualitative analysis in Figure 4. We first visualize the dataset and architecture embeddings in the latent search space in Figure 4 (a). To embed graphs, we combine the graph encoder of Section 3.2 and the graph decoder of Section 3.1 and train it for graph reconstruction on MetaImageNet training tasks. We observe that the architecture whose embeddings are closer to the given dataset achieves better performance on it, compared to architecture embedded far away. For example, the final accuracy of the closest architecture point for the Aircraft dataset is 50.87% while the farthest architecture only achieves 16.77%. In Figure 4 (b), we visualize the latent vectors of the multiple datasets. Similar datasets, such as CIFAR-10 and CIFAR-100 are embedded closely in the latent space, while datasets dissimilar to them, such as MNIST and SVHN, are embedded far away. This enables generating architectures that perform better on each dataset.

4.3 Effectiveness of MetaD2A

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

Generative Abilities as a VAE model We show the basic abilities of VAE where MetaD2A is based. We evaluate generated 10,000 neural architecture samples with the measures Validity, Uniqueness, and Novelty. 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. The result shows MetaD2A can generate 67.31% new graphs that do not belong to the training set and can generate 35.19% various graphs, not picking always-performing-well architecture.

Validity	Uniqueness	Novelty
1.0000	0.3519	0.6731

Table 4: Generative Abilities

Ablation Study on MetaD2A We train different variations of our model on the MetaImageNet, and test on CIFAR10, CIFAR100, and Aircraft in Table 5 with the same experimental setup as the main experiments in Table 2. We observe that combining the performance predictor to the random architecture sampler (Random w/ PP.) enhances the accuracy of the final architecture on all datasets. The MetaD2A generator without the performance predictor (MetaD2A w/o PP.) outperforms the simple random architecture sampler (Random), especially by 15.3% on Aircraft, which demonstrates the effectiveness of MetaD2A over the random sampler. Finally, MetaD2A combined with the performance predictor outperforms all baselines, especially by 22.16% on Aircraft, suggesting that our MetaD2A generator can sample architectures that are more relevant to the given task. We further report the accuracy of the retrieved architecture on CIFAR10 and CIFAR100 with a different number of samples in Figure 5, which shows that our model requires only a small number of samples to obtain a good neural architecture.

Source Dataset	Target Dataset		
MetaImageNet	CIFAR10	CIFAR100	Aircraft
Optimal	94.37	73.51	-
Random	93.52±0.25	70.71±0.53	38.15±0.99
Random w/ PP.	94.02±0.11	72.83±0.48	53.39±3.13
MetaD2A w/o PP.	94.09±0.22	72.76±0.55	53.45±3.27
MetaD2A	**94.37**±0.01	**73.51**±0.00	**60.31**±0.73

Table 5: Ablation study of MetaD2A

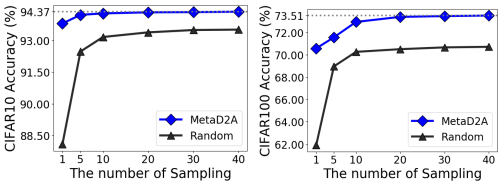


Figure 5: Accuracy over the number of arch. sampling

Ablation Study of The Meta-Performance Predictor and the Set Encoding

We further analyze our performance predictor using the Pearson correlation coefficient on the validation tasks of MetaImageNet to measure the linear correlation between the actual performance and the predicted performance (higher the better). Table 6 shows that using both the dataset and the computational graph of the target architecture as inputs, instead of using one, leads to better performance. Moreover, the proposed encoder yields models with a higher correlation than the mean pooling, which suggests that the higher-order interaction between the input instances is useful for graph generation. Finally, using hierarchical set encoding and bi-directional graph encoding both clearly improve the performance of the predictor.

Input Type		Encoder			Decoder	Pearson
Data	Graph	Mean	Set	Hier.	Bi-dir.	Corr. Coeff.
✓		✓				0.3699
	✓	✓				0.6518
✓	✓	✓				0.7299
✓	✓		✓			0.7647
✓	✓		✓	✓		0.7663
✓	✓		✓	✓	✓	**0.7976**

Table 6: Ablation Study of Predictor.

We also further evaluate the predictor performance for both the Pearson correlation coefficient and the accuracy of the searched architecture on CIFAR10 and CIFAR100 under the same experiment condition of the Table 2. To separate the effect of the predictor, we use random sampling instead of MetaD2A generator for this experiment. Using a larger number of tasks for training improves the performance of the predictor in both measures, suggesting that we can further improve the performance of MetaD2A by collecting more training pairs.

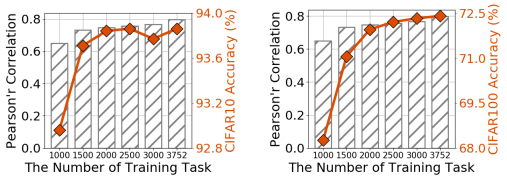


Figure 6: Acc. and Corr. over training task size

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 outperforms conventional NAS methods on various datasets with very small search time (less than a minute) 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.

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.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. *arXiv preprint arXiv:2006.10355*, 2020.

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.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

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.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. *arXiv preprint arXiv:2002.03230*, 2020.

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.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in neural information processing systems (NeurIPS)*, 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)*, 2018a.

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

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. In *Advances in neural information processing systems (NeurIPS)*, 2018b.

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.

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.

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 META DATASET-DEPENDENT ARCHITECTURE GENERATION MODEL

A.1 BUILDING BLOCKS OF THE SET ENCODER

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:

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

where LN and MLP is the layer normalization (Ba et al., 2016) and the multilayer perceptron respectively, and $\boldsymbol{H} \in \mathbb{R}^{n_{\boldsymbol{B}_c} \times \text{d}_{\boldsymbol{H}}}$ is computed with multi-head attention $\text{MH}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$ (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 $\boldsymbol{S} \in \mathbb{R}^{k \times \text{d}_{\boldsymbol{S}}}$ to produce k vectors by slightly modifying $\boldsymbol{H}$ calculation of (9):

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

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):

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

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

A.2 GRAPH DECODING

We use a **GNN-based decoder with asynchronous message passing (GNN-AMP)** which allows message passing to happen only along the *topological order* of the DAGs (Zhang et al., 2019). The graph decoder starts from an initial hidden state $\mathbf{h}_{v_0} = \text{NN}_{\text{init}}(\mathbf{z})$, where NN_{init} is an MLP with $tanh$ output and it finally completes building DAGs by adding a node iteratively. We assume that a node without any predecessors/successors is a start/end node, and if multiple nodes meet aforementioned conditions, we create a virtual starts/end node.

To generate the ith node v_i, we compute the operation type $\text{o}_{v_i} \in \mathbb{R}^{1 \times n_o}$ 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. $\mathbf{h}_{v_i}$ is updated following equation (4) when a new edge is added. During the propagation process of GNN operator, $\mathbf{h}_{v_i}^{(t)}$ is continuously encoded by aggregating the graph-level information. 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. Note that while DVAE (Zhang et al., 2019) demonstrates their performance on the simple task such as graph-to-graph reconstruction, we aim a more challenging problem of generating a DAG from a latent embedding of a dataset.

A.3 BI-DIRECTIONAL GRAPH ENCODING

Graph encoding model is based on **GNN-AMP**. 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 equation (12) by feeding in its predecessors' hidden states $\{u \in \mathcal{V}_{v_i}^{in}\}$.

$$\mathbf{h}_{v_i}^{(t+1)} = \text{UPDATE}(\boldsymbol{y}_{v_i}, \boldsymbol{m}_{v_i}^{(t)}) \text{ , where } \boldsymbol{m}_{v_i}^{(t)} = \sum_{u \in \mathcal{V}_{v_i}^{in}} \text{AGGREGATE}(\mathbf{h}_u^{(t)}) \tag{12}$$

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

$\mathbf{h}_f$. The graph encoding process with **GNN-AMP** can be regarded as a mimic of forwarding propagation of computation flows of neural networks. Inspired by the forward-backward propagation algorithm of neural network, 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 (See Figure 3 (c)). 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.

B 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.

C EXPERIMENTAL SETUP

C.1 DATASET

1) MetaImageNet 447 neural architectures out of the 1,679 tasks used for training MetaD2A generator are searched with GDAS (Dong & Yang, 2019b). We observed that the MetaD2A generator trained with tasks including neural architectures retrieved from GDA tend to converge to generate similar cell structures even for different tasks. Thus, we use additional 1,232 training tasks to increase the exploration diversity with the random search as follows: We first randomly sample five architecture candidates from 1000 architectures, which are in the rank of top-1000 accuracy on the ImageNet-16-120 from NAS-Bench-201 (Dong & Yang, 2020). Then we train them on the dataset of the given task and select the architecture with the highest accuracy among the five. Although architectures searched in this manner are not optimal for the corresponding tasks, we observe that neural architectures generated from the latent space formed through training on those tasks significantly improve the search efficiency compared with the random sampling, as denoted in Table 5. Further, considering collectable database in real-world scenarios are usually not optimal, we believe that this is more realistic approach. To collect tasks for MetaD2A predictor, we randomly sample a subset from the meta-training set/meta-test set of MetaImageNet for meta-training/meta-test, respectively, and a neural architecture from the search space of NAS-Bench201. Then we train the neural architecture on the given subset to get accuracy. In this way, we simply collect 3,752/400 tasks for meta-training/meta-test, respectively. **2) 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. **3) 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. **4) 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. **5) 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. **6) 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. **7) 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.

C.2 BASELINES

We now briefly describe the baseline models and our MetaD2A model. **1) 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. **2) ENAS** (Pham et al., 2018) This is an RL-based NAS method, which introduces the controller to select a sub-graph from a large computational graph, which allows efficient architecture search by sharing parameters among the child models and providing empirical performance as the reward. **3) DARTS** (Liu et al., 2019) This gradient-based NAS method speeds up the search time by introducing differentiable architecture representations with continuous relaxation of the discrete computational graph. We use the first-order DARTS in all experiments. **4) 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. **5) 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. **6) 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`. **7) DrNAS** (Chen et al., 2020) 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**. **8) 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. **9) REA** (Real et al., 2019) This is an evolutional-based search method by using aging based tournament selection, showing evolution can work in NAS. **10) 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. **11) REINFORCE** (Williams, 1992) This is a RL-based NAS. We reward the model with the validation accuracy after 12 epochs of training. **12) 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. **13) DVAE** (Zhang et al., 2019) This uses bayesian optimization to search for the best performing DAGs by using the latent space produced by D-VAE. We use the code at `https://github.com/muhanzhang/D-VAE` **14) FNA** (Fang et al., 2020) This starts from a pretrained seed network and adapts it to a target dataset using a differentiable NAS. When searching and training the architecture, it uses parameter remapping. We make the seed network by retrieving the best-performing architecture from NAS-Bench-201 and train the network on CIFAR-100. The total cost for pre-training is 6856(s). Here we use DARTS for architecture search and we use the same procedure of parameter remapping. **15) 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 6), 7) and 13).

C.3 IMPLEMENTATION DETAILS

Hyperparameter	Value
The number of inputs of class b	20
Dimension of $\mathbf{v}_c$ $\mathrm{d}_{\mathbf{v}_c}$	512
Dimension of $\mathbf{h}_e$ $\mathrm{d}_{\mathbf{h}_e}$	512
Dimension of $\mathbf{S}$ $\mathrm{d}_{\mathbf{S}}$	512
Dimension of $\mathbf{z}$ $\mathrm{d}_{\mathbf{z}}$	56
The number of operators n_o	5
Sampling number n_s for test	1000
Learning rate of generator	1e-4
Learning rate of predictor	1e-4
Batch size of generator	32
Batch size of predictor	32
KL-divergence weighting value λ	5e-3
Training epoch of generator	300
Training epoch of predictor	280

Table 7: Hyperparameter setting of MetaD2A

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. (7). The values of hyperparameters which we used for MetaD2A model in this paper are described in Table 7. For unseen datasets of MetaImageNet tasks, **MNIST**, **SVHN**, **Aircraft**, and **Oxford-IIIT Pets**, we sample $n_s = 1000$ neural architecture candidates by MetaD2A generator, select the best one neural architecture ($N = 1$) with the highest predicted accuracy computed by the meta-performance predictor, and report the mean accuracy of trained the search neural architecture run with 3 different random seeds. For **CIFAR10** and **CIFAR100**, we sample $n_s = 1000$ neural architecture candidates by MetaD2A generator, select $N = 40$ neural architecture candidates in order of high predicted accuracies computed by the meta-performance predictor, retrieve the real accuracies of $N = 40$ neural architecture candidates from NAS-Bench 201 (Dong & Yang, 2020), and report the highest real accuracy among real accuracies as the final performance of the searched neural architecture by MetaD2A. 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.

D MORE RESULTS

D.1 THE EFFECTIVENESS OF SET-DEPENDENT PREDICTOR

To demonstrate the set-dependent predictor on multiple tasks, we compare our preidctor (MetaD2A-GD) with other competitive set-independent predictors such as linaer layer-based, LSTM-based (Hochreiter & Schmidhuber, 1997), GCN-based (Kipf & Welling, 2016) predictors. Note that the proposed predictor can accumulate meta-knowledge from multiple tasks to handle multiple datasets with once training while none of other set-independent predictors can do. Also, we add our predictor with graph encoder (MetaD2A-G) which is set-independent. We train with them on the multiple tasks of MetaImageNet and then compute Pearson correlation coefficient between the predicted accuracy and the true accuracy on 400 unseen MetaImageNet tasks. The higher score means the predictor estimates better the true accuracy. We observe that linear layers-based or LSTM-based predictors fail to be learned on multiple tasks. We conjecture the reason is that they are too simple and ignore the change in accuracy dependent on data even the same graph. GCN-based predictor quite successfully works well even it also set-independent. However, our MetaD2A-DG, which can consider both input dataset and graph, outperforms the GCN predictor. Even MetaD2A-G, which uses graph encoder only performs better than the GCN predictor due to the asynchronous message passing. Our predictor can search architecture for new dataset instantly by designed for capturing both dataset and graph and trained with amortized learning to accumulate meta-knowledge.

	Linear layer	LSTM	GCN	MetaD2A-G	MetaD2A-GD (ours)
Correlation	0.0420	0.0064	0.6076	0.6426	**0.7976**

Table 8: Predictor Comparison

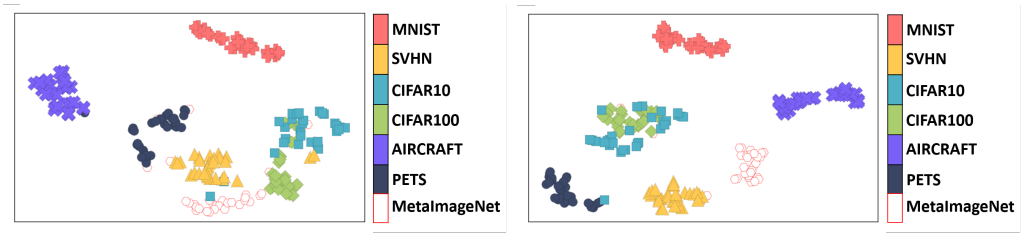
D.2 PERFORMANCE ON FEW-SHOT CLASSIFICATION TASK

We further compared 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 consider. Note that we can generate an architecture for each task almost instantly with MetaD2A for the few-shot task. Following (Elsken et al., 2020; Lian et al., 2019), we adopt bi-level optimization (e.g., MAML framework) to meta-learn initial weight on a meta-training set of mini-imagenet. As shown in the Table 9, the few-shot classification results on MiniImageNet further clearly show the MetaD2A's effectiveness over existing Meta-NAS methods, as well as conventional meta-learning without NAS.

Method	Params (K)	MiniImageNet	
		5-way 1-shot	5-way 5-shot
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 9: Performance on few-shot classification task

D.3 TSNE VISUALIZATION OF Z OF DATASETS



(a) (b)

Figure 7: **(a) and (b)** T-SNE vis. of $\mathbf{z}$ of several datasets. Each marker indicates the subset $\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c\}$ with different seed. We randomly choose the task in MetaImageNet. As the task of MetaImageNet which is constructed from ImageNet can consist of similar classes of other datasets or not, the embedding can be spread over the space as in **(a)** or drawn far from the other datasets' embedding as in **(b)**.

D.4 TSNE VISUALIZATION OF Z OF DATASETS AND GRAPHS

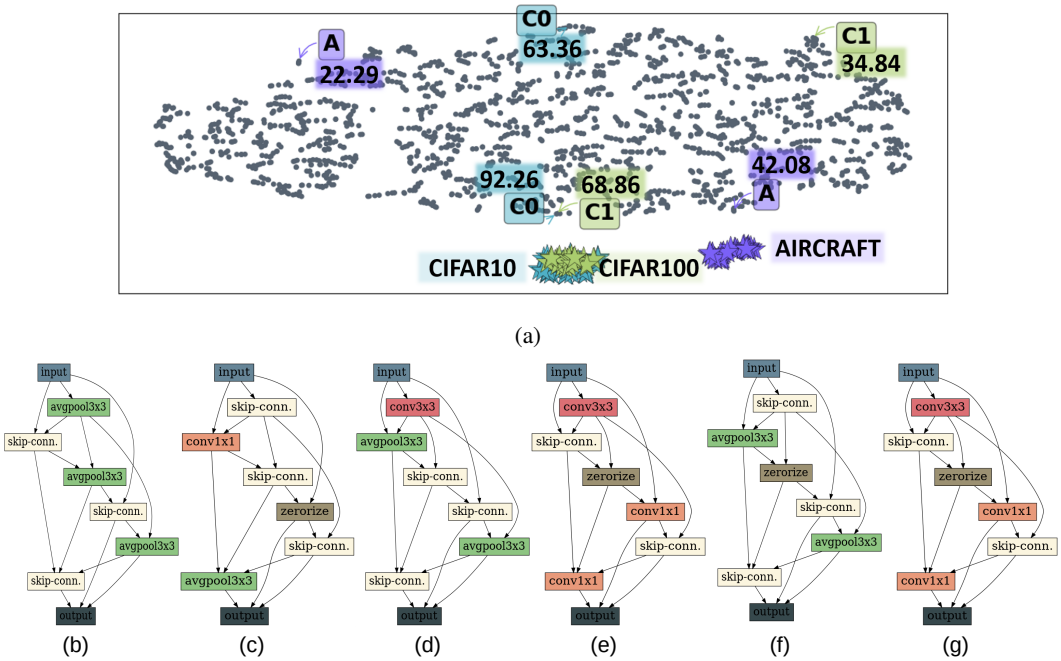


Figure 8: **(a)** T-SNE vis. of Aircraft, CIFAR-10, CIFAR-100 and Graph embeddings. **(b)** The farthest architecture point of AIRCRAFT **(c)** The closest architecture point of AIRCRAFT **(d)** The farthest architecture point of CIFAR-10 **(e)** The closest architecture point of CIFAR-10 **(f)** The farthest architecture point of CIFAR-100 **(g)** The closest architecture point of CIFAR-100

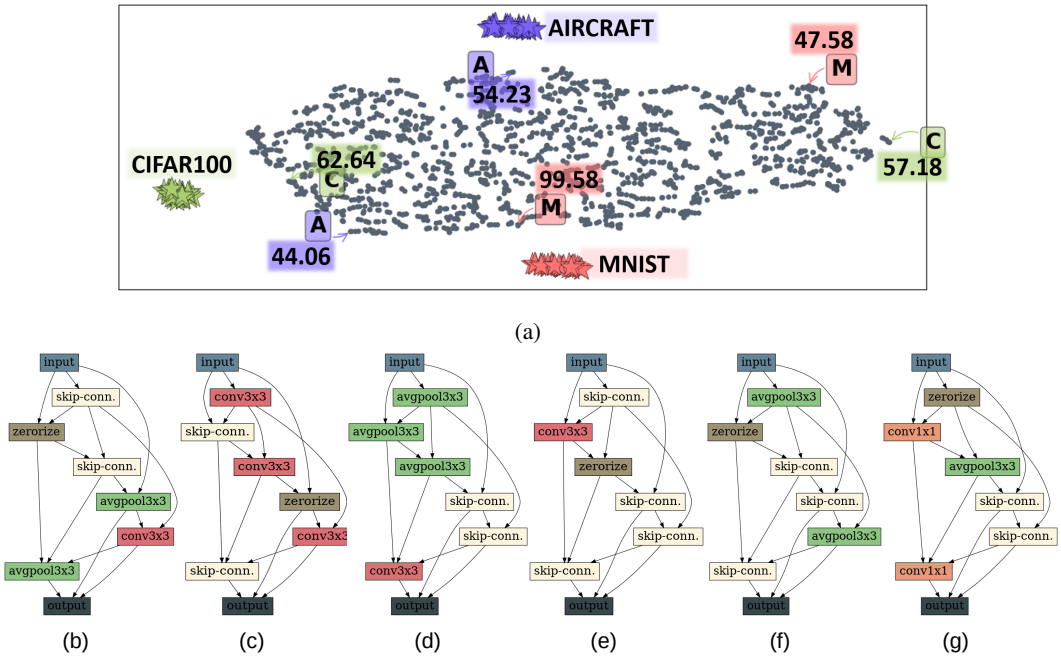


Figure 9: **(a)** T-SNE vis. of Aircraft, CIFAR-100, MNIST and Graph embeddings. **(b)** The farthest architecture point of AIRCRAFT **(c)** The closest architecture point of AIRCRAFT **(d)** The farthest architecture point of CIFAR-100 **(e)** The closest architecture point of CIFAR-100 **(f)** The farthest architecture point of MNIST **(g)** The closest architecture point of MNIST