RAPID NEURAL ARCHITECTURE SEARCH BY LEARNING TO GENERATE GRAPHS FROM DATASETS

Anonymous authors
Paper under double-blind review

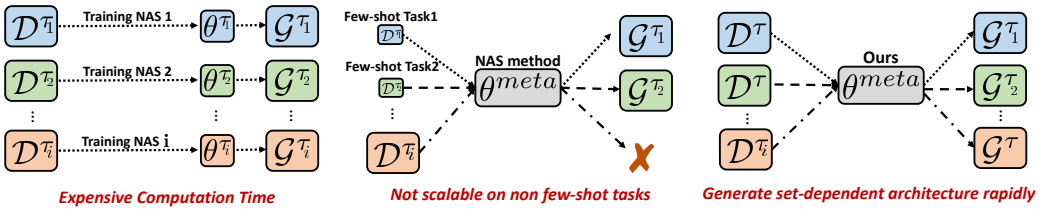
ABSTRACT

Despite the success of Neural Architecture Search (NAS) on various tasks which have shown to output networks that largely outperform manually-designed networks, conventional NAS methods have mostly tackled the optimization of the network architecture for a single task (dataset), which does not generalize well across multiple tasks (datasets). Moreover, since such conventional methods search for an optimal architecture from scratch for every given task, they will incur large computational cost, which will be problematic with limited time and monetary budgets. In this paper, we propose an efficient NAS framework which aims towards significantly reducing the architecture search time for a new task by leveraging prior-knowledge obtained from a database consisting of datasets and pre-trained networks. The proposed MetaD2A (Meta Dataset-to-Architecture) framework is comprised of a set encoder and a graph decoder, and can instantly generate a neural network architecture for a novel dataset, by learning a cross-modal latent space from which we can obtain data-dependent architecture representations for unseen tasks. The experimental results demonstrate that our method meta-learned with MetaD2A on subsets of ImageNet-1K successfully generates high-performance architectures for unseen datasets such as CIFAR-10 and CIFAR-100 with the inference time of less than 0.03 GPU seconds. We believe that the MetaD2A proposes a new research direction for rapid NAS as well as ways to utilize the knowledge from the rich database of models and datasets that have been accumulated over a long period of time.

1 INTRODUCTION

The effective architecture designs have promoted the success of deep learning in many applications (Krizhevsky et al., 2012; He et al., 2016; Szegedy et al.; Vaswani et al., 2017; Zhang et al., 2018; Cho et al., 2014; Huang et al., 2017). However, due to the vast search space, the hand-designed architectures required time-consuming manual trials of human experts. Recently, to tackle such inefficiency in the manual architecture design process, researchers have proposed various *Neural Architecture Search (NAS)* that performs automated architecture search, achieving neural architectures with impressive performances on various tasks that outperforms human-designed counterparts (Baker et al., 2017; Chen et al., 2018; Kandasamy et al., 2018; Liu et al., 2018a; Luo et al., 2018; Pham et al., 2018; Zoph & Le, 2017; Dong & Yang, 2019b; Liu et al., 2019).

While those NAS methods perform well when given a single target task, they generally do not utilize learned information well when searching for a new task. Recently, large benchmark datasets for NAS (NAS-101, NAS-201) (Ying et al., 2019; Dong & Yang, 2020) have been introduced, which provide a large database of architectures and their performances on benchmark datasets. Yet, most conventional NAS methods cannot benefit from the availability of such large datasets, due to their task-specific natures which requires to train the model from scratch repeatedly for each new task (See Figure 1). Thus, searching for an optimal architecture for a new task (dataset) may take a large amount of computation, which may be problematic when the time and monetary budget is limited. To overcome such limitations of conventional NAS system, we want a rapid NAS method that can search for an optimal architecture within a few seconds for a given dataset by leveraging prior-knowledge accumulating over many other tasks.



(a) Task-specific NAS (b) Existing meta-NAS (c) Ours

Figure 1: **Comparison among types of NAS system (a)** Task-specific NAS methods require expensive computation time, which are trained from scratch for each task τ_i. **(b)** While existing meta-NAS works adapt meta-knowledge from several tasks to new tasks, they have difficulty to apply it to non few-shot tasks such as CIFAR10 classification. **(c)** Ours can search task-specific architectures through inference process within one second.

To this end, we propose a generalizable neural architecture search framework which can generate a task-specific architecture given an unseen dataset within a few seconds. Specifically, our model learns a cross-modal latent space that encodes representations for task-dependent architectures using a database which contain datasets and their corresponding optimal-architectures. Given a dataset, our model first encodes it using an order- and size-invariant set encoder (e.g. Set Transformer (Lee et al., 2019a)) and embed it onto the learned cross-modal latent space. Then, from the latent embedding, we use a graph decoder to generate auto-regressively generate the computational graph for the target architecture (Zhang et al., 2019).

Several methods(Lian et al., 2019; Elsken et al., 2020; Shaw et al., 2019) have studied transferability of NAS to unseen datasets with prior knowledge obtained across multiple datasets based on gradient-based bi-level optimization (meta-learning) framework such as MAML (Finn et al., 2017). For this, they alternatively learn architecture and initial-weights with bi-level optimization framework. Since the multiple-unrolling step of gradient requires a quite expensive computation time, they try to bypass it by exploiting parallel computation and approximation of MAML (Nichol et al., 2018). Yet, it is still sub-optimal and they target few-shot classification tasks, which are small scale as shown in (b) of Figure 1. However, they are not scalable to large tasks, since they require costly lookahead gradient steps, and large datasets require a considerably larger number of gradient steps in order to minimize loss for all training instances. On the other hand, our method is scalable to large tasks since tasks are encoded as a low-dimensional latent vector and can obtain the solution using a single feed-forward process with amortized inference.

We meta-learn our model on multiple tasks which consists of subsets of ImageNet-1K and validate it to search for set-adaptive architectures from the NAS-Bench201 search space for unseen tasks. Our model achieves at least *4 times faster spees* to search for architecture even when including the meta-training process, and the searched architectures shows competitive results on the classification task when compared to state-of-the-art NAS baselines. Further, we further validate our model meta-learned on MetaD2A by transferring it to multiple datasets such as MNIST, CIFAR-10, CIFAR-100, Aircrafts, and observe that our model can generalize across multiple datasets while taking only 0.03 GPU second to search for a set-adaptive architecture for each task with our meta-learned architecture generator.

To summarize, our contribution in this work is threefold:

- We propose a NAS framework that search set-specific neural architectures even for unseen tasks by learning set-dependent graph representations in cross-modal latent space across the task distribution with the meta-learning framework.
- For effective meta-learning with such latent space, we construct the set encoder-architecture decoder framework considering set-input problem and directed acyclic graph decoding, which can be breakthrough to the training efficiency and scalability problem of existing NAS methods.
- We validate our model on multiple datasets and we observe that it takes less than one GPU second to search a set-adaptive architecture for each dataset, which significantly outperforms conventional NAS models requiring at least a few GPU hours.

2 RELATED WORK

Neural Architecture Search Neural Architecture Search (NAS) is an automated architecture search process to overcome the sub-optimal problem of hand-designed architecture caused when exploring the extensive search space. NAS researches have been shown quite successful performances on a variety of tasks and can be roughly categorized into reinforcement learning (RL) based methods (Zoph & Le, 2017; Zoph et al., 2018; Pham et al., 2018), evolutionary algorithm (EV) based methods (Real et al., 2019), and gradient-based methods (Liu et al., 2019; Cai et al., 2018; Dong & Yang, 2019b; Chen et al., 2020; Xu et al., 2020).

Most NAS methods have been tackled classification problems and recent works introduce NAS for object detection (Ghiasi et al., 2019), semantic image segmentation (Chen et al., 2018), and generative model (Gong et al., 2019). The most related NAS approach is NAO (Luo et al., 2018), which map DAGs into continuous latent embedding space. While inputs of ours are datasets allowing to generate data-dependent architecture, NAO performs the graph-to-graph reconstruction for a single task. Similar with NAO, most NAS methods have a single task applicable limitation in that it is difficult to transfer trained NAS methods for one dataset to search architectures for other datasets. Even some NAS methods apply the cell searched from small scale dataset to the large-scale dataset by more stacking the cell, it is sub-optimal since the large-scale dataset do not consider during the searching process.

One of the main NAS research streams is to effectively reduce tremendous searching costs (Cai et al., 2018; Liu et al., 2018a; Pham et al., 2018; Liu et al., 2019; Chen et al., 2020), where the recent work (Dong & Yang, 2019b) achieves searching architecture within a few GPU hours. PC-DARTS (Xu et al., 2020) reduces GPU memory overhead and searching time by using partial channel connections, which selects channels performing operations. However, due to the task-specific nature of aforementioned methods, they might be practically inefficient. They should train a model from the scratch for each new unseen task repeatably during at least a few GPU hours. The proposed method can provide task-specific architecture for each unseen task without additional training by leveraging generalized knowledge obtained from other tasks after once meta-training.

Neural Architecture Search with Meta-Learning Meta-learning (learning to learn) aims to train a model to learn new concepts fast leveraging prior knowledge, similar to human education (Vinyals et al., 2016; Snell et al., 2017; Sung et al., 2018; Lee et al., 2019b; Hou et al., 2019; Finn et al., 2017; Nichol et al., 2018). One of the most active research streams is the gradient-based (Finn et al., 2017; Nichol et al., 2018) consisting of the bi-level optimization framework, which usually meta-learn the initial parameters to optimize the objective of the outer loop with task-specific parameters obtained from initial parameters of the inner loop.

Recently, (Elsken et al., 2020; Lian et al., 2019) propose to combine existing the NAS called DARTS (Liu et al., 2019) and the gradient-based meta-learning framework, which are adaptable to different tasks with few-gradient steps by meta-learning operation weighting parameters (architecture parameters) and initial parameters. (Shaw et al., 2019) propose the gradient-based meta-architecture search with Bayesian formulation to learn the agnostic representation over the multiple tasks simultaneously.

While all those methods show promising results when transferred on new tasks, the gradient-based inevitably requires a high computation time due to the expensive multiple unrolling gradient steps for one meta-update of each task. Due to the high computation cost, some are applicable only on small scale tasks such as few-shot classification tasks (Elsken et al., 2020; Lian et al., 2019) or some try to bypass it with the first-order approximation (Lian et al., 2019; Shaw et al., 2019) or use GPU in parallel (Shaw et al., 2019), yet, all of them might be sub-optimal.

In this paper, to tackle the scalability problem, we encode a dataset as a set representation in the low-dimensional latent space regardless of size and exploit fast GNN propagation instead of the expensive gradient update process to generate set-adaptive architecture. In the Table 2, the computation time including both meta-training and meta-testing of ours is extremely fast compared with other baselines allowing to transfer for normal (non few-shot) datasets such as CIFAR10, CIFAR100, CUB.

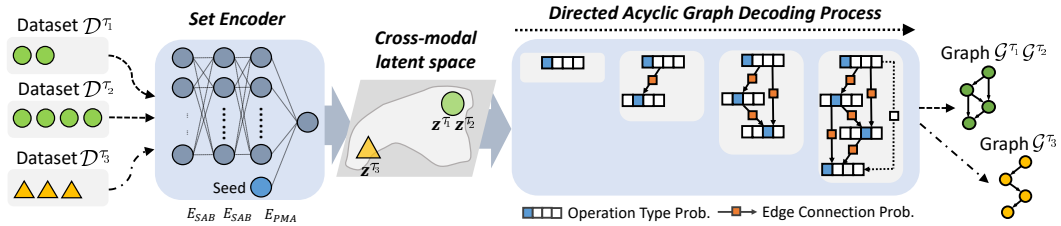


Figure 2: **Overview of our method** We first takes dataset $\mathcal{D}^{\tau_i}$ to encode it as latent vector $\mathbf{z}^{\tau_i}$. From $\mathbf{z}^{\tau_i}$, graph decoder predicts operation types and the edge connection from the root node to the ending node one-by-one updating graph embedding. Finally, we get a set-specific architecture represented as a DAG.

3 METHOD

Our goal is to design the set-dependent architecture sampler which generates high-quality architecture candidates by utilizing the prior knowledge accumulated from the rich database consisting of datasets and its optimized-neural architectures. For this, our model learn to map datasets and corresponding architectures into the cross-modal continuous latent space $\mathcal{Z}$. The proposed model consists of the set encoder for compressing dataset as an informative single latent code and the Graph Neural Networks (GNNs) based directed acyclic graphs (DAGs) decoder. Additionally, we introduce the task-generalized performance predictor, which takes the dataset and the sampled architectures allowing to select the best performing architecture for the dataset.

Assume database are accumulated with many tasks where each task $\tau = \{\mathcal{D}, \mathcal{G}, \boldsymbol{s}\}$ consists of dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{Y}\}$, its set-optimized architecture represented as the DAG $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ and the accuracy $\boldsymbol{s}$ computed from the trained architecture corresponding to $\mathcal{G}$ for the $\mathcal{D}$. $\boldsymbol{X}, \boldsymbol{Y}$ are the set of instances and target labels, respectively. Each node $v \in \mathcal{V}$ has related computational operator (i.e. 3-by-3 convolution, 1-by-1 convolution, mean pooling) and edges $(u, v) \in \mathcal{E}$ represent flows of an output of the operator of u into an input of the operator of v. Then we can formulate the problem for our set-dependent architecture sampler that the DAGs decoder $p_{\boldsymbol{\theta}}(\mathcal{G}|\mathbf{z})$ and the set encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathcal{D})$ learn parameters $\boldsymbol{\theta}, \boldsymbol{\phi}$, respectively generalized over the task distribution $p(\tau)$. Suppose we randomly obtain task τ from task distribution $p(\tau)$ for training the model. After such the training is over, our model with $\boldsymbol{\theta}, \boldsymbol{\phi}$ can easily generate the set-specific neural architecture candidates $\{\mathcal{G}_i^\tau\}_{i=1}^{n_s}$ where n_s is the number of sampling for the unseen dataset $\mathcal{D}^\tau$ by decoding the latent embeddings $\{\mathbf{z}_i^\tau | \mathbf{z}_i^\tau \in \mathcal{Z}\}_{i=1}^{n_s}$ sampled from the prior $p(\mathbf{z})$ consisting of $\boldsymbol{\mu}^\tau$ and $\boldsymbol{\sigma}^\tau$ generated from the set encoder for given $\mathcal{D}^\tau$. The task-generalized performance predictor $p_{\boldsymbol{\omega}}(\boldsymbol{s}|\mathcal{D}, \mathcal{G})$ parameterized with $\boldsymbol{\omega}$ is also trained from the task distribution $p(\tau)$ and then select the best performing architecture $\mathcal{G}^\tau$ of $\{\mathcal{G}_i^\tau\}_{i=1}^{n_s}$ based on predicted accuracy $\{\boldsymbol{s}_i^\tau\}_{i=1}^{n_s}$.

3.1 META SET-DEPENDENT ARCHITECTURE SAMPLER

Dataset Encoding For compressing the entire instances of dataset $\mathcal{D}^\tau$ to a single informative latent code $\mathbf{z}^\tau$, the set encoder should process any size of input set varying for each task and summarize consistent information not depending on the order of multiple instances of the set (permutation-invariance). Practical examples to fulfill those requirements are pooling operations such as mean-pooling or sum-pooling. However, those simple non-parametric aggregation functions are too simple to perform high-order interactions between input elements of the set to extract the relevant information for DAGs generation. Additionally, for encoding set of class sets, those non-hierarchical pooling methods do not consider label information of instances, which is a critical limitation resulting in poor performance (Lee et al., 2020).

Therefore, we introduce a **Hierarchical Set Pooling** which stacks two permutation-invariant encoders with learnable parameters to be trained to extract the related information for generating set-optimized DAGs from the dataset. The first one is a shared intra-class encoder to capture class prototypes which reflects label information and the second one is an inter-class encoder which considers the relationship between class prototypes and aggregates them to encode a single latent vector.

Specifically, for a given dataset $\mathcal{D}^\tau$, we randomly sample sub-set $\{\mathbf{x} | \mathbf{x} \in \boldsymbol{B}_c^\tau\} \in \mathbb{R}^{n_{\boldsymbol{B}_c} \times \mathrm{d_x}}$ from class instance set $\boldsymbol{X}_c^\tau$ and takes it as an input of the intra-class encoder IntraSetPool to encode class prototypes $\mathbf{v}_c \in \mathbb{R}^{1 \times \mathrm{d_{\mathbf{v}_c}}}$ for each class $c = 1, ..., C$. The set of class prototypes $\{\mathbf{v}_c^\tau\}_{c=1}^C$ is an input

of inter-class encoder InterSetPool to generate the set representation $\mathbf{h}_e^\tau \in \mathbb{R}^{1\times \mathrm{d}_{\mathbf{h}_e}}$:

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

Both set encoders have the same structure constructed by two building blocks multiple Set Attention Blocks SAB (Lee et al., 2019a) and Pooling by Multi-head Attention PMA (Lee et al., 2019a). SAB is an attention-based block, which re-creates the features of each element reflecting relations between features of other elements of the set such as (We slightly abuse $\boldsymbol{X}$):

$$\text{SAB}(\boldsymbol{X}) = \text{LN}(\boldsymbol{H}) + \text{MLP}(\boldsymbol{H})) \quad \text{where} \quad \boldsymbol{H} = \text{LN}(\boldsymbol{X} + \text{MH}(\boldsymbol{X}, \boldsymbol{X}, \boldsymbol{X})) \tag{2}$$

where LN and MLP is the layer normalization (Ba et al., 2016) and the multilayer perceptron, respectively and $\boldsymbol{H} \in \mathbb{R}^{n_{\boldsymbol{B}_c}\times \mathrm{d}_{\boldsymbol{H}}}$ is computed with multi-head attention $\text{MH}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$ (Vaswani et al., 2017) which queries, keys, and values are elements of input set $\boldsymbol{X}$ (See Appendix ? for the more details of the multi-head attention). Features encoded from the SAB layers can be pooled by PMA on learnable seed vectors $\boldsymbol{S} \in \mathbb{R}^{k\times \mathrm{d}_{\boldsymbol{S}}}$ to produce k vectors by slightly modifying $\boldsymbol{H}$ calculation of (2):

$$\text{PMA}(\boldsymbol{X}) = \text{LN}(\boldsymbol{H}) + \text{MLP}(\boldsymbol{H})) \quad \text{where} \quad \boldsymbol{H} = \text{LN}(\boldsymbol{X} + \text{MH}(\boldsymbol{S}, \text{MLP}(\boldsymbol{X}), \text{MLP}(\boldsymbol{X}))) \tag{3}$$

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):

$$\text{E}(\{\text{x}|\text{x} \in \pi\boldsymbol{X}\}) = \pi\text{E}(\{\text{x}|\text{x} \in \boldsymbol{X}\}) \tag{4}$$

Since LN, MLP, and MH are row-wise computation functions, SAB and PMA is permutation equivarint by definition (4) (See Appendix ? for the proof). Finally, we stack SAB twice followed by the pooling PMA for our both set encoder. The resultant forms are as follows:

$$\mathbf{v}_c^\tau = \text{IntraSetPool}\big(\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c^\tau\}\big) = \text{PMA}\big(\text{SAB}\big(\text{SAB}\big(\{\mathbf{x}|\mathbf{x} \in \boldsymbol{B}_c^\tau\}\big)\big)\big) \tag{5}$$

$$\mathbf{h}_e^\tau = \text{InterSetPool}\big(\{\mathbf{v}_c^\tau\}_{c=1}^C\big) = \text{PMA}\big(\text{SAB}\big(\text{SAB}\big(\{\mathbf{v}_c^\tau\}_{c=1}^C\big)\big)\big) \tag{6}$$

Then the latent code $\mathbf{z}^\tau \in \mathbb{R}^{1\times \mathrm{d}_{\mathbf{z}}}$ is sampled from a dataset-conditioned Gaussian distribution with diagonal covariance where $\text{NN}_{\boldsymbol{\mu}^\tau}, \text{NN}_{\boldsymbol{\sigma}^\tau}$ are one linear layer:

$$\begin{aligned} \boldsymbol{\mu}^\tau, \boldsymbol{\sigma}^\tau &= \text{NN}_{\boldsymbol{\mu}^\tau}(\mathbf{h}_e^\tau), \text{NN}_{\boldsymbol{\sigma}^\tau}(\mathbf{h}_e^\tau) \\ \mathbf{z}^\tau &\sim q_{\boldsymbol{\phi}}(\mathbf{z}^\tau|\mathcal{D}^\tau) = \mathcal{N}(\boldsymbol{\mu}^\tau, (\boldsymbol{\sigma}^{\tau 2})) \end{aligned} \tag{7}$$

Graph Decoding For modeling DAGs generation from the latent code $\mathbf{z}^\tau$, we construct GNN-based decoder $p_{\boldsymbol{\theta}}(\mathcal{G}^\tau|\mathbf{z}^\tau)$. GNN-based decoder have been actively used to approximate distribution of task which can be represented as graphs such as social networking, protein modeling and molecule design (Liu et al., 2018b; Liao et al., 2019; Jin et al., 2020). However, since those decoders are based on the simultaneous message passing among nodes, which ignores *topological order* of DAGs, using those decoder directly have a critical limitations. Therefore, we exploit **GNN-based decoder with asynchronous message passing (GNN-AMP)** which computes message passing for nodes along the *topological order* of the DAGs (Zhang et al., 2019). Note that while DVAE (Zhang et al., 2019) demonstrates their performance on the simple task such as graph-to-graph reconstruction, we target the task which generates a graph from a latent code encoded from dataset, which is not trivial.

The graph decoder starts from a initial hidden state $\mathbf{h}_{v_0}^\tau \in \mathbb{R}^{1\times \mathrm{d_h}}$ mapped by NN_{init} with $\mathbf{z}^\tau$:

$$\text{h}_{v_0} = \text{NN}_{\text{init}}(\mathbf{z}^\tau) \tag{8}$$

where NN_{init} is MLP with $tanh$. For i^{th} node v_i according to *topological order*, we compute computational operation type probability $\text{o}_{v_i} \in \mathbb{R}^{1\times n_o}$ over n_o operations based on the current graph state represented as the last hidden node $\mathbf{h}_{\mathcal{G}^\tau} := \mathbf{h}_{v_{i-1}}$:

$$\text{o}_{v_i} = \text{NN}_{\text{node}}(\mathbf{h}_{\mathcal{G}^\tau}) \tag{9}$$

where NN_{node} is MLP followed by *softmax*. In the case the predicted v_i type is the ending type, decoding process is stopped and all leaf nodes are connected to node v_i. Otherwise we update hidden state $\mathbf{h}_{v_i}$. The internal hidden state $\mathbf{h}_{v_i}^{(t)}$ for i^{th} node v_i is updated at time step t:

$$\mathbf{h}_{v_i}^{(t+1)} = \text{UPDATE}(\boldsymbol{y}_{v_i}, \boldsymbol{m}_{v_i}^{(t)}) \tag{10}$$

where update function UPDATE is a gated recurrent unit (GRU) (Cho et al., 2014), $\boldsymbol{y}_{v_i}$ is an operation type of v_i, and $\boldsymbol{m}_{v_i}^{(t)}$ is the incoming message to v_i:

$$\boldsymbol{m}_{v_i}^{(t)} = \sum_{u \in \mathcal{V}_{v_i}^{in}} \text{AGGREGATE}(\mathbf{h}_u^{(t)}) \tag{11}$$

$\mathcal{V}_{v_i}^{in}$ is a set of predecessors with incoming edges to v_i and AGGREGATE is the aggregation function consisting of mapping and gating function with MLPs. Assume that a node without any predecessors (successors) is a starting (ending) node. If multiple nodes meet aforementioned conditions, a virtual starting (ending) node is created connecting with the multiple nodes. During the propagation process of GNN operator, $\mathbf{h}_{v_i}^{(t)}$ is continuously encoded by aggregating graph level information. For all previously processed nodes $\{v_j | j = i-1, ..., 1\}$, we decide whether to link directed edge from v_j to v_i by computing edge connection probability $e_{\{v_j, v_i\}}$ and sampling the edge based on $e_{\{v_j, v_i\}}$:

$$e_{\{v_j, v_i\}} = \text{NN}_{\text{edge}}(\mathbf{h}_j, \mathbf{h}_i) \tag{12}$$

where NN_{edge} is MLP followed by $sigmoid$. $\boldsymbol{h}_{v_i}$ is updated following equation (10) when a new edge is added. Following (Zhang et al., 2019), 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.

Meta-training Strategy for neural architecture search We train the model by maximizing the approximated lower bound suitable for each task τ as follows:

$$\max_{\boldsymbol{\phi}, \boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}^\tau \sim q_{\boldsymbol{\phi}}(\mathbf{z}^\tau | \mathcal{D}^\tau)} \big[\log p_{\boldsymbol{\theta}}(\mathcal{G}^\tau | \mathbf{z}^\tau) \big] - \lambda \cdot KL\big[q_{\boldsymbol{\phi}}(\mathbf{z}^\tau | \mathcal{D}^\tau) || p(\mathbf{z}^\tau) \big] \tag{13}$$

where each dimension of prior $p(\mathbf{z})$ factorizes into $\mathcal{N}(0, 1)$. KL-divergence between two multivariate Gaussian distributions is a regularization, which denotes simple closed form (Kingma & Welling, 2014) and λ is the scalar weighting term. We implicitly replace sampling as the reparametrization trick for $\mathbf{z}^\tau$ allowing to train by stochastic gradient descent (Kingma & Welling, 2014). The expected log-likelihood of (13) can be represented as the negative cross-entropy terms for both nodes and edges and we slightly modify it by referring set-optimized architecture $\tilde{\mathcal{G}}^\tau$ such as:

$$L_{\boldsymbol{\phi}, \boldsymbol{\theta}}^\tau(\mathcal{D}^\tau, \tilde{\mathcal{G}}^\tau) = \sum_{v_i \in \mathcal{V}^\tau} \Big\{ L_{CE}^\tau(o_{v_i}, \tilde{o}_{v_i}) + \sum_{v_j \in \mathcal{V}_i^\tau} L_{BCE}^\tau(e_{\{v_j, v_i\}}, \tilde{e}_{\{v_j, v_i\}}) \Big\} \tag{14}$$

where L_{CE}, L_{BCE} is cross-entropy and binary cross-entropy loss, respectively. The reference graph is needed when meta-training phase to accumulate meta-knowledge for NAS. When the meta-testing stage, we can generate a task-adaptive graph by taking only a dataset with meta-trained our NAS system $\boldsymbol{\theta}^*, \boldsymbol{\phi}^*$. Finally, we train the model by minimizing the resultant objective $\mathcal{L}_{\boldsymbol{\phi}, \boldsymbol{\theta}}^\tau(\mathcal{D}^\tau, \tilde{\mathcal{G}}^\tau)$ for our meta-learning purpose:

$$\min_{\boldsymbol{\phi}, \boldsymbol{\theta}} \sum_{\tau \sim p(\tau)} \Big\{ L_{\boldsymbol{\phi}, \boldsymbol{\theta}}^\tau(\mathcal{D}^\tau, \tilde{\mathcal{G}}^\tau) + \lambda \cdot KL\big[q_{\boldsymbol{\phi}}(\mathbf{z}^\tau | \mathcal{D}^\tau) || p(\mathbf{z}^\tau) \big] \Big\} \tag{15}$$

Note that $\boldsymbol{\phi}, \boldsymbol{\theta}$ are generalized across task distribution.

In the meta-test stage, for a given unseen dataset $\hat{\mathcal{D}}$ which is split into training set $\hat{\mathcal{D}}_{tr}$ and test set $\hat{\mathcal{D}}_{te}$, we can generate task-adaptive neural architecture $\mathcal{G}^*$ with meta-trained parameters $\boldsymbol{\phi}^*, \boldsymbol{\theta}^*$ by taking $\hat{\mathcal{D}}_{tr}$ through the feed-forward process. Algorithm 1 and 2 describes our meta-training and meta-testing, respectively.

Algorithm 1 Meta-training

Input: $(\mathcal{D}^1, \tilde{\mathcal{G}}^1), \ldots, (\mathcal{D}^{N_\tau}, \tilde{\mathcal{G}}^{N_{mtr}})$
Input: Pretrained θ, Learning rate α_ϕ, α_θ
Output: ϕ^*, θ^*
Randomly initialize ϕ
while not converged **do**
  for $\tau = 1$ to N **do**
    Randomly sample $\mathcal{B}^\tau \subset \mathcal{D}^\tau$
    Compute $\mathcal{G}^\tau = Dec_{\boldsymbol{\theta}}\big(Enc_{\boldsymbol{\phi}}(\mathcal{B}^\tau)\big)$
    Compute $\mathcal{L}^\tau_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathcal{D}^\tau, \tilde{\mathcal{G}}^\tau)$
  end for
  $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha_\phi \frac{1}{N_\tau} \sum_{\tau=1}^{N_\tau} \mathcal{L}^\tau_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathcal{D}^\tau, \mathcal{G}^\tau)$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha_\theta \frac{1}{N_\tau} \sum_{\tau=1}^{N_\tau} \mathcal{L}^\tau_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathcal{D}^\tau, \mathcal{G}^\tau)$
end while

Algorithm 2 Meta-testing

Input: $\hat{\mathcal{D}}^1, ..., \hat{\mathcal{D}}^{N_{mte}}$
Input: ϕ^*, θ^*
Output: $\mathcal{G}^*$

for $\tau = 1$ to N_{te} **do**
  Randomly sample $\hat{\mathcal{B}}^\tau \subset \hat{\mathcal{D}}^\tau_{tr}$
  Compute $\mathcal{G}^* = Dec_{\boldsymbol{\theta}^*}(Enc_{\boldsymbol{\phi}^*}(\hat{\mathcal{B}}^\tau))$
  Training $\mathcal{G}^*$ on $\hat{\mathcal{D}}_{tr}$ with parameters $\boldsymbol{\pi}$
  Evaluate $\mathcal{G}^*_{\boldsymbol{\pi}^*}$ on $\hat{\mathcal{D}}_{te}$
end for

3.2 TASK-GENERALIZED PERFORMANCE PREDICTOR

The MetaD2A sampler generates set-dependent architecture candidates which are likely to have high-performance. We naturally design additional **task-generalized performance predictor (TGPP)** to select the best performing architecture candidate among sampled architectures, where similar to MetaD2A sampler, TGPP learns to generalization on various tasks by taking the dataset and sampled architectures to predict accuracies. TGPP is constructed with two encoders for processing the dataset and the DAGs as latent vectors, respectively followed linear layers with non-linearity to predict accuracy taking aformentioned two latent vectors.

For the set encoder, we simply use **Hierarchical Set Pooling** of Section 3.1. Graph encoding model is based on **GNN-AMP**. For a given graph candidate $\mathcal{G}^\tau$, we sequentially perform message passing for nodes following a topological order of the DAG $\mathcal{G}^\tau$. We iteratively compute the hidden state of each node v, we update hidden states $\boldsymbol{h}^\tau_v$ using the equation 10 by feeding in its predecessors' hidden states. For starting node v_o which the input is the empty set, we output the zero vector as $\boldsymbol{h}_o$. We use the last hidden states of the ending node as the output $\mathbf{z}^\tau_{\mathcal{G}}$ of the graph encoder. We concatenate the latent vectors $\mathbf{z}^\tau_{\mathcal{D}}$ and $\mathbf{z}^\tau_{\mathcal{G}}$. **Bi-directional encoding** (Zhang et al., 2019) The graph encoding process with AMP can be regard as a mimic of forward propagation of computation flows of neural networks. Similar with backward propagation algorithm, we can reverse node order and perform encoding process. In this case, the final node is the starting points. We perform graph encoding process to generate $\mathbf{z}^\tau_{\mathcal{G}_r}$ and concatenate it to $\mathbf{z}^\tau_{\mathcal{D}}$ and $\mathbf{z}^\tau_{\mathcal{G}}$. and one linear layer with ReLU takes it and predicts the trained accuracy v. We demonstrate the effectiveness of performance predictor in Table ? and ? of Section 4. We train the performance predictor with MSE loss between the predicted accuracy and the true accuracy of each tasks $L^\tau_\omega = (s - \text{PP}(\mathcal{D}^\tau, \mathcal{G}^\tau))^2$.

Table 1: **Performance on unseen MetaD2A** We validate all models for the 20 test tasks. Found architectures are trained for 2 times with the different seeds. We report the mean accuracies and 95% confidence intervals. The bold is the highest performance and the underline is the second one.

Method	Transfer	Search Time (sec)	Expense ($)	MetaD2A Top-1 Accuracy	MetaD2A Top-5 Accuracy
ResNet (He et al., 2016)	N	N/A	xx	xx	xx
ENAS (Pham et al., 2018)	N	247479.12	100.37	28.08±1.53	64.08±1.87
RSPS (Li & Talwalkar, 2019)	N	11009.18	4.46	60.63±1.91	89.47±1.05
DARTS (Liu et al., 2019)	N	9714.14	3.94	43.14±4.65	76.67±3.34
SETN (Dong & Yang, 2019a)	N	30738.71	12.47	61.02±1.86	88.97±0.95
GDAS (Dong & Yang, 2019b)	N	25272.03	10.25	64.35±1.70	89.94±0.90
PC-DARTS (Xu et al., 2020)	N	xx	xx	xx±xx	xx±xx
DrNAS (Chen et al., 2020)	N	10949.95	4.44	67.27±1.30	91.20±0.72
Ours	Y	27.03 / xx.xx	xx	**69.22±1.46**	**91.92±0.82**

4 EXPERIMENT

4.1 SEARCH SPACE

Our method is the cell-based NAS algorithm following the search space of NAS-Bench-201 (Dong & Yang, 2020) consisting of 15,625 possible neural architecture candidates. 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 stride 2 convolution layer for the down-sampling. A fully connected layer is attached to the macro skeleton for classification. Each cell is DAG which consists of 4 nodes with 5 operation candidates such as zerorize, skip connection, 1-by-1 convolution, 3-by-3 convolution, and 3-by-3 average pooling. We add a starting node and an ending node to the cell during training. After cell generation, we delete those two nodes for constructing architectures.

4.2 DATASET

MetaD2A We build MetaD2A from the ImageNet-1K (Deng et al., 2009). We downsample image sizes to 32×32 and split ImageNet-1K into 600/400 classes for meta-training/meta-test, respectively. For all image sub-datasets of tasks, we randomly sample 20 classes with an average of 26K images, each of which in training set only can be built from the first 600 classes, while the sub-datasets of test sets only can be built from the second 400 classes. In the training set, we generate 447 set-specific architectures from the search space of cell-based NAS-Bench-201 (Dong & Yang, 2020) with GDAS (Dong & Yang, 2019b) software. We fix 20 tasks randomly sampled for the testing set.

To validate the transferability of our model after once trained on MetaD2A, we meta-test our model on multiple datasets such as MNIST, CIFAR-10, CIFAR-100, Aircraft. Details of each dataset are as follows. **MNIST**: This is a standard image classification dataset which contains 70K 28×28 grey colored images in 10 classes. We scale images up to 32×32 to compare all the baseline properly. The original dataset is split into a training set with 60K images and a test set with 10K images. **CIFAR-10** This dataset consists of 10 classes with 32×32 colour images. The training set contains 5K images for each classes and total 50K images. The test set consists of 1K images for each class and total 10K images. **CIFAR-100** This contains the same images with **CIFAR-10** and divide them as 100 fine-grained classes. Each class has 500/100 images for training set and test set, respectively. **Aircraft** (Maji et al., 2013) It is fine-grained classification benchmark dataset containing 30 different aircraft classes with 10000 images. We resize all images as 32×32. Following (Dong & Yang, 2020), we randomly split all the original test set into two groups of the same size and assign the first group to the validation set and the other to the test set.

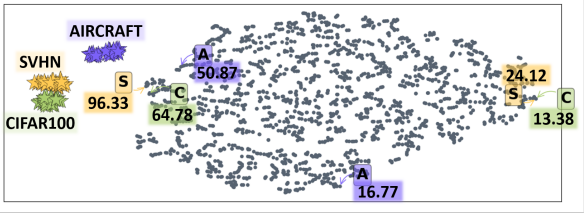
4.3 TRAINING DETAILS

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. Note that 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-testing. We pre-train the graph decoder on the graph-to-graph reconstruction tasks by leveraging rich 15,625 architectures provided from NAS-Bench-201 (Dong & Yang, 2020). Even any graph encoder can be used for this tasks, we exploit the encoder of Zhang et al. (2019). We use mini-batch gradient descent to train the model with Eq. (15) (Jin et al., 2018).

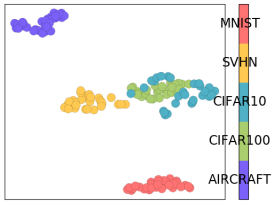
The dimensions of latent vector $\mathbf{z}^{\tau}$ and hidden state $\boldsymbol{h}$ is 56 and 501, respectively. The number of operation type n_o is 5. For each dataset, we randomly sample $n_{\mathcal{B}}$=20 images from each classes

Table 2: **Performance on multiple datasets** We transfer ours to target datasets after training on MetaD2A, while other baselines are trained for each dataset from scratch. Note that it does mean ours only requires inference time but the others require the whole search time. We report the mean accuracies and 95% confidence intervals over 2 runs. We search architecture 2 times with different seed for each model and found architectures are trained for 2 times with the different seeds.The bold is the highest performance and the underline is the second one.

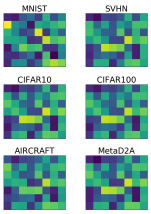
Method	Source Dataset	Search Time (sec)[†]	Target Dataset MNIST	SVHN	CIFAR-10	CIFAR-100	Aircraft
Optimal	-	-	-	-	94.37	73.51	-
ResNet	N/A	N/A	99.62±0.16	xx.xx±x.xx	93.97±0.00	70.86±0.00	47.77±2.71
REA	N/A	0.02 / x.xx	-	-	93.92±0.30	71.84±0.99	-
RS	N/A	0.01 / xx.xx	-	-	93.70±0.36	71.04±1.07	-
REINFORCE	N/A	0.12 / xx.xx	-	-	93.85±0.37	71.71±1.09	-
BOHB	N/A	3.59 / xx.xx	-	-	93.61±0.52	70.85±1.28	-
DVAE	N/A	xx.xx	-	-	xx.xx	xx.xx	-
RSPS	N/A	8007.13	99.68±0.06	96.15±0.17	84.07±3.61	52.31±5.77	42.32±5.16
DARTS	N/A	11625.77	98.77±0.90	65.62±33.60	54.30±0.00	15.61±0.00	22.53±0.03
SETN	N/A	34139.53	99.70±0.03	95.71±0.80	87.64±0.00	59.05±0.24	44.62±5.44
GDAS	N/A	31609.80	99.64±0.05	95.68±0.58	93.61±0.09	70.70±0.30	53.74±0.64
PC-DARTS	N/A	xx.xx	xx.xx	xx.xx	xx.xx	xx.xx	xx.xx
DrNAS	N/A	22447.00	99.57±0.01	96.34±0.01	93.76±0.00	71.5±0.75	40.84±0.94
Ours	MetaD2A	22.47	**99.68**±0.00	**96.57**±0.59	**94.37**±0.01	**73.51**±0.00	**60.41**±3.05



(a) T-SNE vis. of Aircraft, SVHN, CIFAR-100 and Graph embeddings



(b) T-SNE vis. of $\mathbf{z}$



(c) $\mathbf{z}$ Heatmap

Figure 5: **(a)** T-SNE visualization of $\mathbf{z}$ from a dataset and graphs. In Appendix A, we include more examples and corresponding found architectures. **(c)** T-SNE visualization of $\mathbf{z}$ from several datasets. **(d)** Heatmap of $\mathbf{z}$ from several datasets.

and total number of input images is multiplication between class size and the number of images per class $n_C \times 20$. The batch size is 32. The λ of (13) is 2.5e-3. We train our model during 600 epoch for MetaD2A training dataset and use the model of 300 epoch for meta-test phase. We use 1e-3 and 1e-4 for the set encoder and the graph decoder, respectively. We follows hyperparameters of NAS-Bench-201 (Dong & Yang, 2020) for training searched architectures on datasets. Additionally, for all comparison models, we exploit codes provided by NAS-Bench-201 (Dong & Yang, 2020). All results of 200 epoch are reported for all data sets except MNIST. For MNIST, we use the results of 50 epoch.

4.4 Baselines

RSPS (Li & Talwalkar, 2019) This is a combination of random search and weight sharing, which trains randomly sampled sub-graphs from weight shared DAG of the search space. The best performing sub-graph is selected as the final architecture.

ENAS (Pham et al., 2018) ENAS is RL-based NAS method, which introduces the controller to choose sub-graph from a large computational graph and allows efficient search of architecture by sharing parameters among the child models for providing empirical performance as the reward.

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

DARTS (Liu et al., 2019) This gradient-based NAS method speeds up the search time by introducing differentiable architecture representations through continuous relaxation. We adopt first-order DARTS in all experiments.

SETN (Dong & Yang, 2019a) SETN is an one-shot NAS method, where selectively samples competitive child candidates by learning quality evaluation of candidates based on the validation loss.

GDAS (Dong & Yang, 2019b) This proposes Gumbel-Softmax based differentiable sampler learned to minimize validation loss with the architecture sampled from DAGs. This method can search the architecture for CIFAR-10 within four GPU hours.

DrNAS (Chen et al., 2020) For enhancing generalization performance of differential architecture search, DrNAS introduces Dirichlet distribution allowing to approximate architecture distribution.

PC-DARTS (Xu et al., 2020) This model is a gradient-based NAS partially sampling channels to apply operations, resulting in efficiency improvement of GPU memory and search time compared to DARTS.

Ours This is our model as described in the section 3, allowing real-time neural architecture search based on set-dependent latent vector.

4.5 Search Results

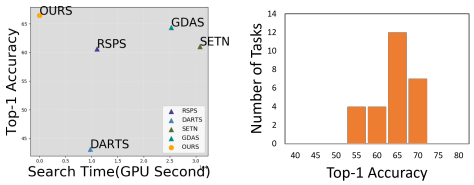


Figure 6

In Table 1, we validate whether our model meta-learned from training set of MetaD2A can search set-specific architectures for unseen datasets of MetaD2A. Note that for the unseen MetaD2A tasks, all baseline models need training processes for each dataset to search set-specific architectures, while our model can generate set-dependent architectures through inference process. All searched architectures are trained from the scratch for classification problems on each corresponding dataset during 200 epochs. We report mean accuracies over 20 tasks with two different seeds such as 20×2=40 tasks. We get search times for each baseline models by averaging over the searching times of architectures for the 20 unseen tasks. For our model, we report both searching time (0.0053s) averaging on the 20 unseen tasks and meta-training time (2291.36s). Since Ours-RS uses trained our graph decoder, we regard Ours-RS has the same meta-training time with Ours. We calculate the expense by using cost, 1.46$/h, of Tesla P100 on Google Cloud. All models are run on Nvidia 2080ti GPU.

The proposed method outperforms all baselines taking an extremely faster search time (0.0053s). Even including meta-training, our method is at least 4× faster, and only considering meta-testing, it is 1.8M×. Our method could be useful when constructing a practical system that searches set-specific architectures rapidly for given datasets with very low-cost $\$2 \times 10^{-6}$ for each. In the Figure **??**, we observe that even Ours-RS which is set-independent model works well. We conjecture that the latent space of the model is well-formed to produce an architecture that is likely to have high accuracy. It is consistent with the results that the tendency of accuracy rankings of architectures among datasets are similar (Dong & Yang, 2020). We can interpret that our model map those tendency into the continuous space. Additionally, when we use set-dependent latent vectors, we can search better set-adaptive architectures as shown in Figure **??** (b).

In the Table 2, we validate that our model trained from source dataset MetaD2A is generalized for multiple datasets such as MNIST, SVHN, CIFAR-10, CIFAR-100, and Aircraft. We compute the mean search time for 2 runs of our model on the CIFAR-10. Our model obtains task-specific architecture through feed-forward pass within rapid inference time 0.03(s) yielding decent accuracies, while other methods require expensive task-specific training time for architecture search.

4.6 Qualitative Analysis

We perform several qualitative analysis in the Figure 5. We first project latent vectors encoded from a dataset or graphs with T-SNE into the metric space as shown in the (a) and (b) of the Figure 5. For encoding graphs, we train the graph encoder of (Zhang et al., 2019) with our decoder. The architecture generated from the graph representation which is close to the set ones tend to obtain

higher accuracy than the architecture far away in the metric space. We further draw metric space (c) of latent vectors for multiple datasets. The representations of datasets have different displacements which induce dissimilar architectures generation. Different heatmaps of $\mathbf{z}$ for multiple datasets are consistent with (c). You can see this tendency in Figure **??** (a).

4.7 EFFECTIVENESS OF LEGEND

Source Dataset MetaD2A	Target Dataset CIFAR10	CIFAR100
Optimal	94.37	73.51
Random	93.52±0.25	70.71±0.53
Random w/ PP.	94.02±0.11	72.83±0.48
LEGEND w/o PP.	94.09±0.22	72.76±0.55
LEGEND	**94.37**±0.01	**73.51**±0.00

Table 3: Ablation study

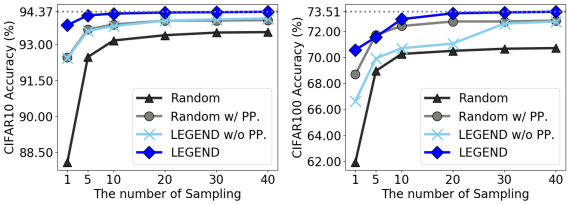


Figure 7:

Input Type Data	Graph	Encoder SetTrans	Hier.	Decoder Bi-dir.	Pearson'r Corr.
✓					0.3699
	✓				0.6518
✓	✓				0.7299
✓	✓	✓			0.7647
✓	✓	✓	✓		0.7663
✓	✓	✓	✓	✓	**0.7976**

Table 4: Abl. study

Training Task size	Pearson'r Corr.
1000	0.6496
1500	0.7312
2000	0.7476
2500	0.7565
3000	0.7669
3752	**0.7976**

Table 5:

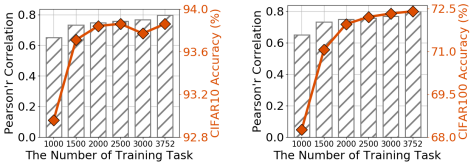


Figure 8:

5 CONCLUSION

We propose the practical framework that successfully searches architecture for a new dataset within one second generalized across data-dependent graph latent distribution. Specifically, we encode the dataset for each task considering set-input problems and exploit it to decode a direct acyclic graph which is a subgraph of a large computational graph of search space. Latent embeddings in low-dimensional space used in our model accelerate searching time and allows scale to large datasets. Our model significantly outperforms conventional NAS methods on search time when validated on various datasets such as MetaD2A, MNIST, SVHN, CIFAR-10, CIFAR-100, and Aircraft. We hope that our model is a meaningful step towards a practical NAS system for real-world scenarios.

REFERENCES

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.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in neural information processing systems*, pp. 8699–8710, 2018.

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 *CVPR09*, 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*, pp. 3681–3690, 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. *arXiv preprint arXiv:2001.00326*, 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/CVF Conference on Computer Vision and Pattern Recognition*, 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.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7036–7045, 2019.

Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3224–3234, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Ruibing Hou, Hong Chang, MA Bingpeng, Shiguang Shan, and Xilin Chen. Cross attention network for few-shot classification. In *Advances in Neural Information Processing Systems*, pp. 4003–4014, 2019.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

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.

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

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

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. 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*, pp. 10657–10665, 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 *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*, pp. 7795–7804, 2018b.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pp. 7816–7827, 2018.

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

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

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*, 2019.

Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *Advances in Neural Information Processing Systems*, pp. 11227–11237, 2019.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pp. 4077–4087, 2017.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1199–1208, 2018.

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*, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

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*, 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*, 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*, 2018.

A APPENDIX

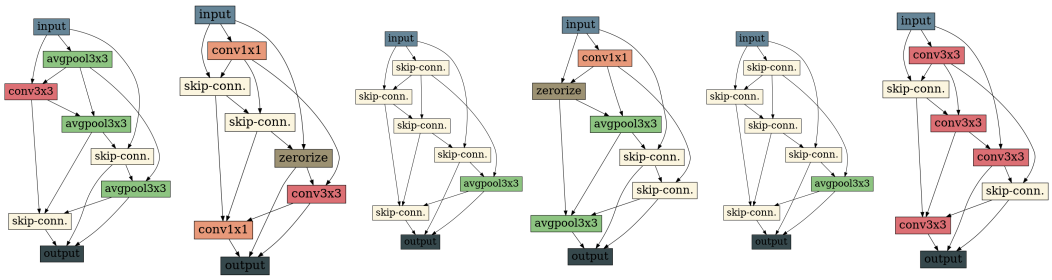
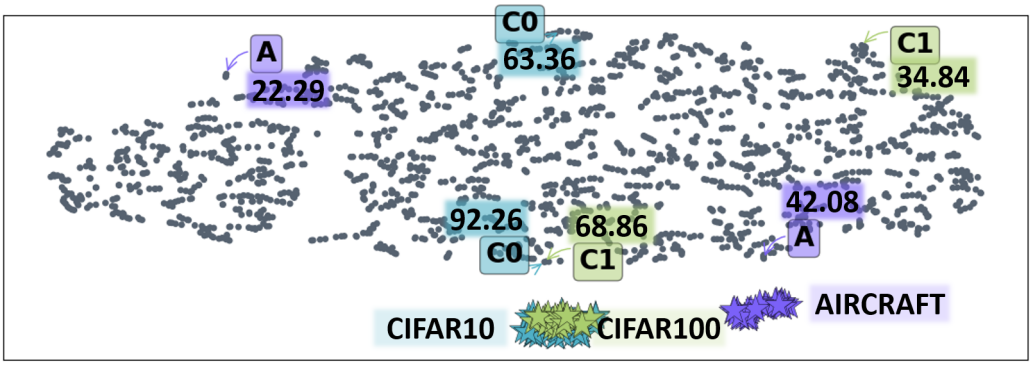


Figure A: **(a)** The farthest architecture point of AIRCRAFT **(b)** The closest architecture point of AIRCRAFT **(c)** The farthest architecture point of CIFAR-100 **(d)** The closest architecture point of CIFAR-100 **(e)** The farthest architecture point of SVHN **(f)** The closest architecture point of SVHN



(a) T-SNE vis. of Aircraft, CIFAR-10, CIFAR-100 and Graph embeddings

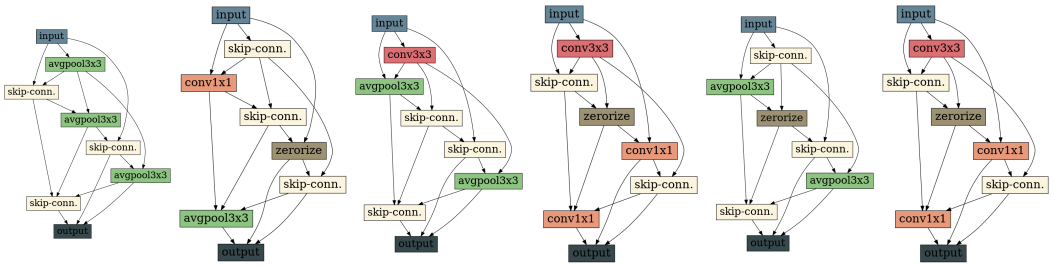


Figure C: **(a)** The farthest architecture point of AIRCRAFT **(b)** The closest architecture point of AIRCRAFT **(c)** The farthest architecture point of CIFAR-10 **(d)** The closest architecture point of CIFAR-10 **(e)** The farthest architecture point of CIFAR-100 **(f)** The closest architecture point of CIFAR-100