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). 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 when the budget and time is limited.

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 model zoo consisting of datasets and pretrained networks. The proposed framework is a composition of a set encoder and a graph decoder, which learns data-dependent architecture generative representations in the cross-modal latent space, that is transferrable to 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 the proposed approach opens up the possibilities of practical and rapid NAS frameworks which allow to utilize the knowledge from the rich database of models and datasets that have been accumulated over the recent years.

1 Introduction

The effective architecture design have been promoted the great 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). Due to the numerous search space, the hand-designed architectures requires time-consuming manual tries of human experts. In recent, researchers have proposed the Neural Architecture Search (NAS) which is a automated architecture search algorithm showing quite successful performance on various tasks (Baker et al., 2017; Chen et al., 2018; Kan-dasamy et al., 2018; Liu et al., 2018; Luo et al., 2018; Pham et al., 2018; Zoph & Le, 2017; Dong & Yang, 2019b; Liu et al., 2019).

While those NAS methods do well in a single target task, they have a difficulty to utilize learned information for a new task. However, people may need a practical NAS system that searches architectures within a few seconds for a given dataset by leveraging prior-knowledge accumulating over many other tasks. Recently, even NAS benchmark datasets (NAS-101, NAS-201) (Ying et al., 2019; Dong & Yang, 2020) are introduced which provide fluent architectures and corresponding accuracies on benchmark datasets with reproducible codes allowing fair comparison, most conventional NAS methods are difficult to leverage them due to their task-specific approaches. They train the model from scratch repeatedly for each new task with considerable computation time and resources such as (a) of Figure 1.

In this paper, we propose a generalized architecture search approach that takes a unseen dataset and generates a set-tailored architecture within a few seconds. Our model learns cross-modal latent representations across a rich task distribution consisting of datasets and corresponding optimal-architecture pairs. We should consider a set-input problem to construct the set encoder, which outputs an informative latent code regardless of the order of input instances and the size of the...
Under review as a conference paper at ICLR 2021

Expensive Computation Time

Training NAS 1

…

Training NAS 2

Training NAS

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 \( \tau_i \). (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.

dataset. For this, we exploit an attention-based set transformer (Lee et al., 2019a). From the latent embedding, we generate architecture auto-regressively based on the graph neural networks (Zhang et al., 2019). As shown in Figure 1 compared with few-shot meta-NAS methods which accompany expensive multiple gradient computations/update for entire instances of given tasks, our method is scalable to large tasks since tasks are encoded as a low-dimensional latent vector through the feed-forward process.

Several methods (Lian et al., 2019; Elsken et al., 2020; Shaw et al., 2019) have studied transferability of NAS for 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.

We meta-learn our model on several tasks of MetaD2A which consist of subsets of ImageNet-1K and validate it to search set-adaptive architectures from the NAS-Bench201 search space for unseen tasks. Our model achieves at least 4 times faster to search architecture, including the meta-training process, and the searched architectures trained from the scratch shows competitive results on the classification task when compared to other baselines. Further, we validate our model meta-learned on MetaD2A by transferring it on multiple datasets such as MNIST, CIFAR-10, CIFAR-100, Aircrafts. We observe that our model can generalize across multiple datasets and takes only 0.03 GPU second to search for a set-adaptive architecture for each task since the property of architecture generation through the inference process accelerates the time effectively.

To summarize, our contribution in this work is threefold:

- We propose NAS system 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

2.1 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).

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., 2018; Pham et al., 2018; Liu et al., 2019), where the recent work (Dong & Yang, 2019b) achieves searching architecture within a few GPU hours. 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 repeatedly 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.

2.2 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.

3 BACKGROUND

We first review the two major ingredients of our model: set-input problem and graph neural network (GNN) with asynchronous message passing scheme.

Set-input Problem: The definition of set-input problem is that input are instances in a set and its target is a label for the set. The model to solve the set-input problem should satisfy the two conditions:
1) permutation-invariance, where informative set embedding of the set should be consistent not depending on the order of elements in the set. 2) processing any size of input. Representative simple example to fulfill those requirements is pooling operations such as mean or sum by aggregating features transformed from instances of the dataset with neural networks. 

Zaheer et al. (2017) have been proven that aforementioned two criteria can be met by stacking permutation-equivariant layer $E$ which satisfies below condition for any permutation $\pi$ on a set $\mathcal{X} \in \mathbb{R}^{n \times d}$:

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

(1)

Lee et al. (2019a) design Set Transformer which an attention-based model for set-input problem by building multiple set attention blocks $E_{SAB}$:

$$E_{SAB}(\mathcal{X}) = rN(\mathcal{H}) + rM(\mathcal{H})$$

where $\mathcal{H}$ is computed with multi-head attention $rA(Q, K, V)$ (Vaswani et al., 2017) which queries, keys, values are $\mathcal{X}$. Since $rN$, $rM$, and $rA$ are row-wise computation functions, $E_{SAB}$ is permutation equivariant by definition (1). Features encoded from stack of $E$ layers can be pooled by multi-head attention $E_{PMA}$ on a learnable seed vector $s \in \mathbb{R}^{1 \times d}$ by slightly modifying $\mathcal{H}$ calculation of (2):

$$\mathcal{H} = rN(\mathcal{X} + rA(s, rM(\mathcal{X}), rM(\mathcal{X})))$$

(3)

The resultant permutation invariant form to generate set representation $z \in \mathbb{R}^{1 \times d}$:

$$z = E_{PMA}(E_{SAB}(\mathcal{X}))$$

(4)

**GNN with asynchronous message passing:** Zhang et al. (2019) introduces a graph neural network for directed acyclic graph (DAG) $G$ consisting of a collection of nodes and edges $(\mathcal{V}, \mathcal{E})$, which computes message passing for nodes along a topological order of the DAG. The internal hidden state $h_{v_i}^{(t)}$ for $i$th node $v_i$ is updated at time step $t$:

$$h_{v_i}^{(t+1)} = U(y_{v_i}, m_{v_i}^{(t)})$$

(5)

where update function $U$ is a gated recurrent unit (GRU) (Cho et al., 2014), $y_{v_i}$ is a type of $v_i$, and $m_{v_i}^{(t)}$ is the incoming message to $v_i$:

$$m_{v_i}^{(t)} = \sum_{u \in \mathcal{V}_{v_i}} M(h_{u}^{(t)})$$

(6)

$\mathcal{V}_{v_i}$ is a set of predecessors with incoming edges to $v_i$ and $M$ is 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, $h_{v_i}^{(t)}$ is continuously encoded by aggregating graph level information.

4 **Method**

The proposed method is a composition of the model for the set-input problem and GNN operator for DAGs, which are the dataset encoder and the architecture decoder parameterized by $\phi$ and $\theta$, respectively. Assume the dataset $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$ where $\mathcal{X}, \mathcal{Y}$ are the set of instances and target labels, respectively and corresponding architecture represented as the DAG $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where each node $v \in \mathcal{V}$ has related computational operator and edges $(u, v) \in \mathcal{E}$ represent flows of an output of $u$ into an input of $v$. Our model learns to map a dataset and corresponding architecture into the cross-modal latent space $\mathcal{Z}$.

For training the model through meta-learning framework, suppose we randomly obtain task $\tau$ consisting of the dataset $D^\tau$ and corresponding set-optimized neural architecture $G^\tau$ from task distribution $p(\tau)$. Then we can formulate the problem that the DAGs decoder $p_{\theta}(G|z)$ and the set encoder $q_{\phi}(z|D)$ learn meta-parameters $\theta, \phi$, respectively generalized over the task distribution $p(\tau)$. After such the meta-learning is over, our model with $\theta, \phi$ can easily generate the set-specific neural architecture $G^\tau$ for the unseen dataset $D^\tau$ by decoding the latent embedding $z^\tau$ sampled from the prior $p(z)$ consisting of $\mu^\tau$ and $\sigma^\tau$ generated from encoding given $D^\tau$. 

Under review as a conference paper at ICLR 2021
We first take dataset $D^τ$ to encode it as latent vector $z^τ$. From $z^τ$, 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.

### 4.1 Dataset Encoding

We can define the dataset encoding $q_φ(z^τ|D^τ) := Enc_φ(D^τ)$ as the set-input problem. Even though fixed poolings such as sum or mean will practically work well, we adopt a parameterized Set Transformer with the equation (4) allowing high-order interaction between input elements to extract useful information for architecture generation. For a given dataset $D^τ$, we first compose sub-set of instances $B^τ ∈ \mathbb{R}^{n_B \times d_i}$ by randomly sampling $n_B$ instances from each class $C$ of $X^τ$ (where $n$ is $n_B \times n_C$ and $d_i$ is the size of instances) and take it as an input of the encoder.

$$h^τ_{enc} = E_{PMA}(E_{SAB}(E_{SAB}(B^τ)))$$

Then the latent code $z^τ ∈ \mathbb{R}^{1 \times d_z}$ is sampled from a dataset-conditioned Gaussian distribution with diagonal covariance:

$$\mu^τ, \sigma^τ = NN_{μτ}(h^τ_{enc}), NN_{στ}(h^τ_{enc})$$

$$z^τ ∼ q_φ(z^τ|D^τ) = N(μ^τ, diag(σ^τ^2))$$

where $NN_{μτ}, NN_{στ}$ are one-layer MLP.

### 4.2 Graph Decoding

The graph decoder $p_θ(G^τ|z^τ) := Dec_θ(z^τ)$ starts from a initial hidden state $h_{v_0}^τ ∈ \mathbb{R}^{1 \times d_h}$ mapped by $NN_{init}$ with $z^τ$:

$$h_{v_0} = NN_{init}(z^τ)$$

where $NN_{init}$ is MLP with tanh. For $i^{th}$ node $v_i$ according to topological order, we compute computational operation type probability $o_{v_i} ∈ \mathbb{R}^{1 \times n_o}$ over $n_o$ operations based on the current graph state represented as the last hidden node $h_G := h_{v_{i-1}}$:

$$o_{v_i} = NN_{node}(h_G)$$

where $NN_{node}$ is MLP followed by softmax. In 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 $h_{v_i}$ using equation (5) and continue the decoding process. 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 $e_{\{v_j, v_i\}}$:

$$e_{\{v_j, v_i\}} = NN_{edge}(h_j, h_i)$$
Note that φ

In the meta-test stage, for a given unseen dataset 

meta-testing, respectively.

taking our meta-learning purpose:

where each dimension of prior θ

stage, we can generate a task-adaptive graph by taking only a dataset with meta-trained our NAS

is needed when meta-training phase to accumulate meta-knowledge for NAS. When the meta-testing

where

late Gaussian distributions is a regularization, which denotes simple closed form (Kingma & Welling,

trick for

and edges and we slightly modify it by referring set-optimized architecture

expected log-likelihood of (12) can be represented as the negative cross-entropy terms for both nodes

end for

where NN_{edge} is MLP followed by sigmoid, \( h_v \) is updated following equation (5) when a new

e 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

dge 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.

4.3 Meta-training Strategy for Neural Architecture Search

We train the model by maximizing the approximated lower bound suitable for each task \( \tau \) as follows:

\[
\max_{\phi, \theta} \mathbb{E}_{z^\tau \sim q_{\phi}(z^\tau|D^\tau)} \left[ \log p_{\theta}(G^\tau|z^\tau) \right] - \lambda \cdot KL[q_{\phi}(z^\tau|D^\tau)||p(z^\tau)]
\]

(12)

where each dimension of prior \( p(z) \) factorizes into \( N(0,1) \). KL-divergence between two multivariate

Gaussian distributions is a regularization, which denotes simple closed form (Kingma & Welling,

2014) and \( \lambda \) is the scalar weighting term. We implicitly replace sampling as the reparametrization

trick for \( z^\tau \) allowing to train by stochastic gradient descent (Kingma & Welling, 2014). The expected

log-likelihood of (12) can be represented as the negative cross-entropy terms for both nodes

and edges and we slightly modify it by referring set-optimized architecture \( G^\tau \) such as:

\[
L_{\phi, \theta}^\tau(D^\tau, \hat{G}^\tau) = \sum_{v_i \in V^\tau} \left\{ L_{CE}^\tau(o_{v_i}, \hat{o}_{v_i}) + \sum_{v_j \in V^\tau_j} L_{BCE}^\tau(c_{v_j, v_i}, \hat{c}_{v_j, v_i}) \right\}
\]

(13)

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 \( \theta^*, \phi^* \). Finally, we train the model by minimizing the resultant objective \( L_{\phi, \theta}^\tau(D^\tau, \hat{G}^\tau) \) for

our meta-learning purpose:

\[
\min_{\phi, \theta} \sum_{\tau \sim \rho(\tau)} \left\{ L_{\phi, \theta}^\tau(D^\tau, \hat{G}^\tau) + \lambda \cdot KL[q_{\phi}(z^\tau|D^\tau)||p(z^\tau)] \right\}
\]

(14)

Note that \( \phi, \theta \) are generalized across task distribution.

In the meta-test stage, for a given unseen dataset \( \hat{D} \) which is split into training set \( \hat{D}_{tr} \) and test set

\( \hat{D}_{te} \), we can generate task-adaptive neural architecture \( G^\tau \) with meta-trained parameters \( \phi^*, \theta^* \) by

taking \( \hat{D}_{tr} \) through the feed-forward process. Algorithm 1 and 2 describes our meta-training and

meta-testing, respectively.
5 Experiment

5.1 Search Space

Our method is the cell-based NAS algorithm following the search space of NAS-Bench-201 \cite{dong2020nasbench201} 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 \cite{he2016deep} 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.

5.2 Dataset

**MetaD2A** We build MetaD2A from the ImageNet-1K \cite{deng2009imagenet}. 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 \cite{dong2020nasbench201} with GDAS \cite{dong2019gdas} 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** \cite{maji2013aircraft} It is fine-grained classification benchmark dataset containing 30 different aircraft classes with 10000 images. We resize all images as 32 × 32. Following \cite{dong2020nasbench201}, 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.

5.3 Training Details

We adopt the teacher forcing training strategy \cite{jin2018search}, 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 \cite{dong2020nasbench201}. Even any graph encoder can be used for this tasks, we exploit the encoder of Zhang et al. \cite{zhang2019graph}. We use mini-batch gradient descent to train the model with Eq. (14) \cite{jin2018search}. The dimensions of latent vector $z$ and hidden state $h$ is 56 and 501, respectively. The number of operation type $n_o$ is 5. For each dataset, we randomly sample $n_B=20$ images from each classes 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 $\lambda$ of (12) is $2.5 \times 10^{-3}$. We train our model during 600 epoch for MetaD2A training dataset and use the model of 300 epoch for meta-test phase. We use $1 \times 10^{-3}$ and $1 \times 10^{-4}$ for the set encoder and the graph decoder, respectively. We follows hyperparameters of NAS-Bench-201 \cite{dong2020nasbench201} for training searched architectures on datasets. Additionally, for all comparison models, we exploit codes provided by NAS-Bench-201 \cite{dong2020nasbench201}. All results of 200 epoch are reported for all data sets except MNIST. For MNIST, we use the results of 50 epoch.
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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Transfer</th>
<th>Search Time (sec)</th>
<th>Expense ($)</th>
<th>MetaD2A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Top-1 Accuracy</td>
</tr>
<tr>
<td>ENAS (Pham et al. 2018)</td>
<td>N</td>
<td>247479.12</td>
<td>100.37</td>
<td>28.08±1.53</td>
</tr>
<tr>
<td>DARTS (Liu et al. 2019)</td>
<td>N</td>
<td>9714.14</td>
<td>3.94</td>
<td>43.14±4.65</td>
</tr>
<tr>
<td>SETN (Dong &amp; Yang 2019a)</td>
<td>N</td>
<td>30738.71</td>
<td>12.47</td>
<td>61.02±1.86</td>
</tr>
<tr>
<td>GDAS (Dong &amp; Yang 2019b)</td>
<td>N</td>
<td>25272.03</td>
<td>10.25</td>
<td>64.35±1.70</td>
</tr>
<tr>
<td>RSPS (Li &amp; Talwalkar 2020)</td>
<td>N</td>
<td>11009.18</td>
<td>4.46</td>
<td>60.63±1.91</td>
</tr>
<tr>
<td>Ours-RS</td>
<td>Y</td>
<td>0.1694 / 2291.36</td>
<td>6×10⁻⁶</td>
<td>63.97±1.87</td>
</tr>
<tr>
<td>Ours</td>
<td>Y</td>
<td>0.0053 / 2291.36</td>
<td>2×10⁻⁶</td>
<td>66.44±1.37</td>
</tr>
</tbody>
</table>

Figure 3: (a) Time vs. Acc. (b) Params. vs. Acc.  Figure 4: Acc. distribution of 20 MetaD2A tasks.

5.4 Baselines

**RSPS** (Li & Talwalkar 2020) 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.

**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.

**Ours-RS** We randomly sample latent vector $z$ from Gaussian distribution and decode it with our trained decoder. This model is independent with the datasets.

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

5.5 Search Results

In Table, 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 $20\times2=40$ 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
Table 2: Performance on multiple datasets We transfer ours to target datasets after training on MetaD2A while baselines are trained for each dataset from scratch. 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Source Dataset</th>
<th>Search Time (sec)</th>
<th>MNIST</th>
<th>SVHN</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>Aircraft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>N/A</td>
<td>81607.13</td>
<td>97.68 ± 0.00</td>
<td>96.15 ± 0.17</td>
<td>73.37 ± 1.60</td>
<td>73.51 ± 0.07</td>
<td></td>
</tr>
<tr>
<td>RSPS</td>
<td>N/A</td>
<td>11625.77</td>
<td>98.77 ± 0.00</td>
<td>65.62 ± 3.30</td>
<td>54.30 ± 0.00</td>
<td>15.61 ± 0.00</td>
<td>22.53 ± 0.03</td>
</tr>
<tr>
<td>DARTS</td>
<td>N/A</td>
<td>34139.53</td>
<td>99.70 ± 0.03</td>
<td>95.71 ± 0.80</td>
<td>87.64 ± 0.00</td>
<td>59.05 ± 0.24</td>
<td>44.62 ± 5.44</td>
</tr>
<tr>
<td>SETN</td>
<td>N/A</td>
<td>31609.80</td>
<td>99.64 ± 0.05</td>
<td>95.68 ± 0.58</td>
<td>93.61 ± 0.09</td>
<td>70.70 ± 0.30</td>
<td>53.74 ± 0.64</td>
</tr>
<tr>
<td>Ours-RS</td>
<td>MetaD2A</td>
<td>0.1694</td>
<td>99.46 ± 0.24</td>
<td>57.48 ± 42.62</td>
<td>86.14 ± 6.90</td>
<td>62.11 ± 3.72</td>
<td>50.89 ± 1.63</td>
</tr>
<tr>
<td>Ours</td>
<td>MetaD2A</td>
<td>0.03</td>
<td>99.62 ± 0.16</td>
<td>96.18 ± 0.16</td>
<td>93.61 ± 0.19</td>
<td>70.74 ± 0.04</td>
<td>53.15 ± 0.82</td>
</tr>
</tbody>
</table>

Figure 5: (a) and (b) T-SNE visualization of $z$ from a dataset and graphs (c) T-SNE visualization of $z$ from several datasets. (d) Heatmap of $z$ from several datasets.

The proposed method outperforms all baselines taking an extremely faster search time (0.0053s). Even including meta-training, our method is at least $4 \times$ faster, and only considering meta-testing, it is $1.8M \times$. You can see this tendency in Figure 3 (a). 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 4, 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 4 (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.

5.6 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

\footnote{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.}
which induce dissimilar architectures generation. Different heatmaps of $z$ for multiple datasets are consistent with (c).

6 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


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


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


A Appendix

You may include other additional sections here.