Task-Adaptive Neural Network Search with Meta-Contrastive Learning

Wonyong Jeong\textsuperscript{1,2,∗†}, Hayeon Lee\textsuperscript{1,2,∗†}, Geon Park\textsuperscript{1,2,∗†}, Eunyoung Hyung\textsuperscript{1,2,†}, Jinheon Baek\textsuperscript{†}, Sung Ju Hwang\textsuperscript{1,2,†}

KAIST\textsuperscript{1}, AITRICS\textsuperscript{2}, Seoul, South Korea
wyjeong@kaist.ac.kr, hayeon926@kaist.ac.kr, geon.park@kaist.ac.kr, ey0301.hyung@samsung.com, jinheon.baek@kaist.ac.kr, sjhwang82@kaist.ac.kr

Abstract

Most conventional Neural Architecture Search (NAS) approaches are limited in that they only generate architectures without searching for the optimal parameters. While some NAS methods handle this issue by utilizing a supernet trained on a large-scale dataset such as ImageNet, they may be suboptimal if the target tasks are highly dissimilar from the dataset the supernet is trained on. To address such limitations, we introduce a novel problem of Neural Network Search (NNS), whose goal is to search for the optimal pretrained network for a novel dataset and constraints (e.g. number of parameters), from a model zoo. Then, we propose a novel framework to tackle the problem, namely Task-Adaptive Neural Network Search (TANS). Given a model-zoo that consists of network pretrained on diverse datasets, we use a novel amortized meta-learning framework to learn a cross-modal latent space with contrastive loss, to maximize the similarity between a dataset and a high-performing network on it, and minimize the similarity between irrelevant dataset-network pairs. We validate the effectiveness and efficiency of our method on ten real-world datasets, against existing NAS/AutoML baselines. The results show that our method instantly retrieves networks that outperform models obtained with the baselines with significantly fewer training steps to reach the target performance, thus minimizing the total cost of obtaining a task-optimal network. Our code and the model-zoo are available at https://github.com/wyjeong/TANS.

1 Introduction

Neural Architecture Search (NAS) aims to automate the design process of network architectures by searching for high-performing architectures with RL [76, 77], evolutionary algorithms [43, 11], parameter sharing [6, 42], or surrogate schemes [38], to overcome the excessive cost of trial-and-error approaches with the manual design of neural architectures [47, 23, 27]. Despite their success, existing NAS methods suffer from several limitations, which hinder their applicability to practical scenarios. First of all, the search for the optimal architectures usually requires a large amount of computation, which can take multiple GPU hours or even days to finish. This excessive computation cost makes it difficult to efficiently obtain an optimal architecture for a novel dataset. Secondly, most NAS approaches only search for optimal architectures, without the consideration of their parameter values. Thus, they require extra computations and time for training on the new task, in addition to the architecture search cost, which is already excessively high.

For this reason, supernet-based methods [8, 37] that search for a sub-network (subnet) from a network pretrained on large-scale data, are attracting more popularity as it eliminates the need for additional

\*Equal contribution.
\†This work was done while the author was interning at AITRICS.

training. However, this approach may be suboptimal when we want to find the subnet for a dataset that is largely different from the source dataset the supernet is trained on (e.g. medical images or defect detection for semiconductors). This is a common limitation of existing NAS approaches, although the problem did not receive much attention due to the consideration of only a few datasets in the NAS benchmarks (See Figure 1, left). However, in real-world scenarios, NAS approaches should search over diverse datasets with heterogeneous distributions, and thus it is important to task-adaptively search for the architecture and parameter values for a given dataset. Recently, MetaD2A [31] has utilized meta-learning to learn common knowledge for NAS across tasks, to rapidly adapt to unseen tasks. However it does not consider parameters for the searched architecture, and thus still requires additional training on unseen datasets.

Given such limitations of NAS and meta-NAS methods, we introduce a novel problem of Neural Network Search (NNS), whose goal is to search for the optimal pretrained networks for a given dataset and conditions (e.g. number of parameters). To tackle the problem, we propose a novel and extremely efficient task-adaptive neural network retrieval framework that searches for the optimal neural network with both the architecture and the parameters for a given task, based on cross-modal retrieval. In other words, instead of searching for an optimal architecture from scratch or taking a sub-network from a single super-net, we retrieve the most optimal network for a given dataset in a task-adaptive manner (See Figure 1, right), by searching through the model zoo that contains neural networks pretrained on diverse datasets. We first start with the construction of the model zoo, by pretraining state-of-the-art architectures on diverse real-world datasets.

Then, we train our retrieval model via amortized meta-learning of a cross-modal latent space with a contrastive learning objective. Specifically, we encode each dataset with a set encoder and obtain functional and topological embeddings of a network, such that a dataset is encoded closer to the network that performs well on it while minimizing the similarity between irrelevant dataset-network pairs. The learning process is further guided by a performance predictor, which predicts the model’s performance on a given dataset.

The proposed Task-Adaptive Network Search (TANS) largely outperforms conventional NAS/AutoML methods (See Figure 2), while significantly reducing the search time. This is because the retrieval of a trained network can be done instantly without any additional architecture search cost, and retrieving a task-relevant network will further reduce the fine-tuning cost. To evaluate the proposed TANS, we first demonstrate the sample-efficiency of our model zoo construction method, over construction with random sampling of dataset-network pairs. Then, we show that the TANS can adaptively retrieve the best-fitted models for an unseen dataset. Finally, we show that our method significantly outperforms baseline NAS/AutoML methods on real-world datasets (Figure 2), with incomparably smaller computational cost to reach the target performance. In sum, our main contributions are as follows:

- We consider a novel problem of Neural Network Search, whose goal is to search for the optimal network for a given task, including both the architecture and the parameters.
- We propose a novel cross-modal retrieval framework to retrieve a pretrained network from the model zoo for a given task via amortized meta-learning with contrastive objective.
- We propose an efficient model-zoo construction method to construct an effective database of dataset-architecture pairs, considering both the model performance and task diversity.
- We train and validate TANS on a newly collected large-scale database, on which our method outperforms all NAS/AutoML baselines with almost no architecture search cost and significantly fewer fine-tuning steps.
2 Related Work

Neural Architecture Search Neural Architecture Search (NAS), which aims to automate the design of neural architectures, is an active topic of research nowadays. Earlier NAS methods use non-differentiable search techniques based on RL [76, 77] or evolutionary algorithms [43, 11]. However, their excessive computational requirements [44] in the search process limit their practical applicability to resource-limited settings. To tackle this challenge, one-shot methods share the parameters [42, 6, 35, 65] among architectures, which reduces the search cost by orders of magnitude. The surrogate scheme predicts the performance of architectures without directly training them [38, 75, 54], which also cuts down the search cost. Latent space-based NAS methods [38, 54, 67] learn latent embeddings of the architectures to reconstruct them for a specific task. Recently, supernet-based approaches, such as OFA [8], receive the most attention due to their high performances. OFA generates the subnet with its parameters by splitting the trained supernet. While this eliminates the need for costly re-training of each searched architecture from scratch, but, it only trains a fixed-supernet on a single dataset (ImageNet-1K), which limits their effectiveness on diverse tasks that are largely different from the training set. Whereas our TANS task-adaptively retrieves a trained neural network from a database of networks with varying architectures trained on diverse datasets.

Meta-Learning The goal of meta-learning [55] is to learn a model to generalize over the distribution of tasks, instead of instances from a single task, such that a meta-learner trained across multiple tasks can rapidly adapt to a novel task. While most meta-learning methods consider few-shot classification with a fixed architecture [56, 20, 48, 40, 33, 30], there are a few recent studies that couple NAS with meta-learning [46, 34, 17] to search for the well-fitted architecture for the given task. However, these NAS approaches are limited to small-scale tasks due to the cost of roll-out gradient steps. To tackle this issue, MetaD2A [31] proposes to generate task-dependent architectures with amortized meta-learning, but does not consider parameters for the searched architecture, and thus requires additional cost of training it on unseen datasets. To overcome these limitations, our method retrieves the best-fitted architecture with its parameters for the target task, by learning a cross-modal latent space for dataset-network pairs with amortized meta-learning.

Neural Retrieval Neural retrieval aims to search for and return the best-fitted item for the given query, by learning to embed items in a latent space with a neural network. Such approaches can be broadly classified into models for image retrieval [21, 14, 66] and text retrieval [73, 9, 63]. Cross-modal retrieval approaches [32, 74, 58] handle retrieval across different modalities of data (e.g. image and text), by learning a common representation space to measure the similarity across the instances from different modalities. To our knowledge, none of the existing works is directly related to our approach that performs cross-modal retrieval of neural networks given datasets.

3 Methodology

We first define the Neural Network Search (NNS) problem and propose a meta-contrastive learning framework to learn a cross-modal retrieval space. We then describe the structural details of the query and model encoders, and an efficient model-zoo construction strategy.

3.1 Problem Definition

Our goal in NNS is to search for an optimal network (with both architectures and parameters) for a dataset and constraints, by learning a cross-modal latent space over the dataset-network pairs. We first describe the task-adaptive network retrieval with amortized meta-contrastive learning.

3.1.1 Meta-Training

We assume that we have a database of neural networks (model zoo) pretrained over a distribution of tasks $p(\tau)$ with each task $\tau = \{D^\tau, M^\tau, s^\tau\}$, where $D^\tau$ denotes a dataset, $M^\tau$ denotes a neural network (or model) trained on the dataset, and $s^\tau$ denotes a set of auxiliary constraints for the network to be found (e.g. number of parameters or the inference latency). Then, our goal is to learn a cross-modal latent space for the dataset-network pairs $(D^\tau, M^\tau)$ while considering the constraints $s^\tau$ over the task distribution $p(\tau)$, as illustrated in Figure 1. In this way, a meta-trained model from diverse $(D^\tau, M^\tau)$ pairs, will rapidly generalize on an unseen dataset $\tilde{D}$; $D^\tau \cap \tilde{D} = \emptyset$ by retrieving a well-fitted neural network on the dataset that satisfies the conditions $s^\tau$.  

3
Real-world Networks

With the above ingredients, we minimize the meta-contrastive learning loss over a task distribution

We can learn such a cross-modal latent space of dataset-network pairs for rapid retrieval by directly

We then introduce

where

The goal of task-adaptive retrieval is to find an appropriate network

obtained from other tasks

pairs:

metric space. Note that, similar to this, we also contrast the query with its corresponding model:

where

Dataset

Real-world Datasets

Illustration for overall framework of our proposed method (TANS): We first construct our model-zoo with pareto-optimal pairs of dataset and network, rather than exhaustively train all possible pairs. We then embed a model and a dataset with a graph-functional model and a set encoder. After that, we meta-learn the cross-modal retrieval network over multiple model-query pairs, guided by our performance predictor.

Task-Adaptive Neural Network Retrieval To learn a cross-modal latent space for dataset-network pairs over a task distribution, we first introduce a novel task-adaptive neural network retrieval problem. The goal of task-adaptive retrieval is to find an appropriate network

task

To this end, we need to calculate the similarity between the dataset-network pair

and

where

is a query (dataset) encoder, 

is a model encoder, which are parameterized with the parameter

and

respectively, and

is a scoring function for the query-model pair. In this way, we can construct the cross-modal latent space for dataset-network pairs over the task distribution with equation 1, and use this space to rapidly retrieve the well-fitted neural network in response to the unseen query dataset.

We can learn such a cross-modal latent space of dataset-network pairs for rapid retrieval by directly solving for the above objective, with the assumption that we have the query and the model encoder: 

. However, we further propose a contrastive loss to maximize the similarity between a dataset and a network that obtains high performance on it in the learned latent space, and minimize the similarity between irrelevant dataset-network pairs, inspired by Faghri et al. [19], Engilberge et al. [18]. While existing works such as Faghri et al. [19], Engilberge et al. [18] target image-to-text retrieval, we tackle the problem of cross-modal retrieval across datasets and networks, which is a nontrivial problem as it requires task-level meta-learning.

Retrieval with Meta-Contrastive Learning Our meta-contrastive learning objective for each task

consisting of a dataset-model pair

aims to maximize the similarity between positive pairs:

while minimizing the similarity between negative pairs:

where

is obtained from the sampled target task

and

is obtained from other targets

, which is illustrated in Figure 3. This meta-contrastive learning loss can be formally defined as follows:

We then introduce

for the meta-contrastive learning:

where

is a margin hyper-parameter and the score function

is the cosine similarity. The contrastive loss promotes the positive

embedding pair to be close together, while at most margin

closer than the negative

embedding pairs in the learned cross-modal metric space. Note that, similar to this, we also contrast the query with its corresponding model:

which we describe in the supplementary material in detail.

With the above ingredients, we minimize the meta-contrastive learning loss over a task distribution

defined with the model

and query

contrastive losses, as follows:

Figure 3: Illustration for overall framework of our proposed method (TANS): We first construct our model-zoo with pareto-optimal pairs of dataset and network, rather than exhaustively train all possible pairs. We then embed a model and a dataset with a graph-functional model and a set encoder. After that, we meta-learn the cross-modal retrieval network over multiple model-query pairs, guided by our performance predictor.
Meta-Performance Surrogate Model  We propose the meta-performance surrogate model to predict the performance on an unseen dataset without directly training on it, which is highly practical in real-world scenarios since it is expensive to iteratively train models for every dataset to measure the performance. Thus, we meta-train a performance surrogate model \( a = S(\tau; \psi) \) over a distribution of tasks \( p(\tau) \) on the model-zoo database. This model not only accurately predicts the performance \( a \) of a network \( M^* \) on an unseen dataset \( D^* \), but also guides the learning of the cross-modal retrieval space, thus embedding a neural network closer to datasets that it performs well on.

Specifically, the proposed surrogate model \( S \) takes a query embedding \( q^\tau \) and a model embedding \( m^\tau \) as an input for the given task \( \tau \), and then forwards them to predict the accuracy of the model for the query. We train this performance predictor \( S(\tau; \psi) \) to minimize the mean-square error loss \( L_s(\tau; \psi) = (s_{\text{acc}}^\tau - S(\tau; \psi))^2 \) between the predicted accuracy \( S(\tau; \psi) \) and the true accuracy \( s_{\text{acc}}^\tau \) for the model on each task \( \tau \), which is sampled from the task distribution \( p(\tau) \). Then, we combine this objective with retrieval objective in equation 4 to train the entire framework as follows:

\[
\min_{\phi, \theta, \psi} \sum_{\tau \in p(\tau)} L_m(\tau; \theta, \phi) + L_q(\tau; \theta, \phi) + \lambda \cdot L_s(\tau; \psi),
\]

where \( \lambda \) is a hyper-parameter for weighting losses.

3.1.2 Meta-Test

By leveraging the meta-learned cross-modal retrieval space, we can instantly retrieve the best-fitted pretrained network \( M \in M \), given an unseen query dataset \( \tilde{D} \in Q \), which is disjoint from the meta-training dataset \( D \in Q \). Equipped with meta-training components described in the previous subsection, we now describe the details of our model at inference time, which includes the following: amortized inference, performance prediction, and task- and constraints-adaptive initialization.

Amortized Inference  Most existing NAS methods are slow as they require several GPU hours for training, to find the optimal architecture for a dataset \( \tilde{D} \). Contrarily, the proposed Task-Adaptive Network Search (TANS) only requires a single forward pass per dataset, to obtain a query embedding \( \tilde{q} \) for the unseen dataset using the query encoder \( Q(\tilde{D}; \theta^\ast) \) with the meta-trained parameters \( \theta^\ast \), since we train our model with amortized meta-learning over a distribution of tasks \( p(\tau) \). After obtaining the query embedding, we retrieve the best-fitted network \( M^\ast \) for the query on the similarity:

\[
M^\ast = \max_{M^\ast} \{ f_{\text{sim}}(\tilde{q}, m^\tau) \mid \tau \in p(\tau) \},
\]

where a set of model embeddings \( \{ m^\tau \mid \tau \in p(\tau) \} \) is pre-computed by the meta-trained model encoder \( E_M(M^\ast; \phi^\ast) \).

Performance Prediction  While we achieve the competitive performance on unseen dataset only with the previously defined model, we also use the meta-learned performance predictor \( S \) to select the best performing one among top \( K \) candidate networks \( \{ M_i \}_{i=1}^K \) based on their predicted performances. Since this surrogate model with module to consider datasets is meta-learned over the distribution of tasks \( p(\tau) \), we predict the performance on an unseen dataset \( \tilde{D} \) without training on it. This is different from the conventional surrogate models [38, 75, 54] that additionally need to train on an unseen dataset from scratch, to predict the performance on it.

Task-adaptive Initialization  Given an unseen dataset, the proposed TANS can retrieve the network trained on a training dataset that is highly similar to the unseen query dataset from the model zoo (See Figure 4). Therefore, fine-tuning time of the trained network for the unseen target dataset \( \tilde{D} \) is effectively reduced since the retrieved network \( M \) has task-relevant initial parameters that are already trained on a similar dataset. If we need to further consider constraints \( s \), such as parameters and FLOPs, then we can easily check if the retrieved models meet the specific constraints by sorting them in the descending order of their scores, and then selecting the constraint-fitted best accuracy model.

3.2 Encoding Datasets and Networks

Query Encoder  The goal of the proposed query encoder \( E_Q(D; \theta) : Q \to \mathbb{R}^d \) is to embed a dataset \( D \) as a single query vector \( q \) onto the cross-modal latent space. Since each dataset \( D \) consists of \( n \) data instances \( D = \{X_i\}_{i=1}^n \in Q \), we need to fulfill the permutation-invariance condition over
the data instances $X_i$, to output a consistent representation regardless of the order of its instances. To satisfy this condition, we first individually transform $n$ randomly sampled instances for the dataset $D$ with a continuous learnable function $\rho$, and then apply a pooling operation to obtain the query vector $q = \sum_{X_i \in D} \rho(X_i)$, adopting Zaheer et al. [69].

**Model Encoder**  
To encode a neural network $M^\tau$, we consider both its architecture and the model parameters trained on the dataset $D^\tau$ for each task $\tau$. Thus, we propose to generate a model embedding with two encoding functions: 1) topological encoding and 2) functional encoding.

Following Cai et al. [8], we first obtain a topological embedding $v^\tau_i$ with auxiliary information about the architecture topology, such as the numbers of layers, channel expansion ratios, and kernel sizes. Then, our next goal is to encode the trained model parameters for the given task, to further consider parameters on the neural architecture. However, a major problem here is that directly encoding millions of parameters into a vector is highly challenging and inefficient. To this end, we use functional embedding, which embeds a network solely based on its input-output pairs. This operation generates the embedding of trained networks, by feeding a fixed Gaussian random noise into each network $M^\tau$, and then obtaining an output $v^\tau_i$ of it. The intuition behind the functional embedding is straightforward: since networks with different architectures and parameters comprise different functions, we can produce different outputs for the same input.

With the two encoding functions, the proposed model encoder generates the model representation by concatenating the topology and function embeddings $[v^\tau_i, v^\tau_i]$, and then transforming the concatenated vector with a non-linear function $\sigma$ as follows: 

$$m^\tau = \sigma([v^\tau_i, v^\tau_i]) = E_M(M^\tau; \phi).$$

Note that, the two encoding functions satisfy the injectiveness property under certain conditions, which helps with the accurate retrieval of the embedded elements in a condensed continuous latent space. We provide the proof of the injectiveness of the two encoding functions in Section C of the supplementary file.

### 3.3 Model-Zoo Construction

Given a set of datasets $D = \{D_1, \ldots, D_K\}$ and a set of architectures $M = \{M_1, \ldots, M_N\}$, the most straightforward way to construct a model zoo $Z$, is by training all architectures on all datasets, which will yield a model zoo $Z$ that contains $N \times K$ pretrained networks. However, we may further reduce the construction cost by collecting $P$ dataset-model pairs, $\{D_i, M_j\}^P_{i=1}$, where $P \ll N \times K$, by randomly sampling an architecture $M \in M$ and then training it on $D \in D$. Although this works well in practice (see Figure 8 (Bottom)), we further propose an efficient algorithm to construct it in a more sample-efficient manner, by skipping evaluation of dataset-model pairs that are certainly worse than others in all aspects (memory consumption, latency, and test accuracy). We start with an initial model zoo $Z_{(0)}$ that contains a small amount of randomly selected pairs and its test accuracies. Then, at each iteration $t$, among the set of candidates $C(t)$, we find a pair $\{D_i, M_j\}$ that can expand the currently known set of all Pareto-optimal pairs w.r.t. all conditions (memory, latency, and test accuracy on the dataset $D$), based on the amount of the Pareto front expansion estimated by $f_{zoo}(\cdot; Z(t))$:

$$\{D_i, M_j\}_{(t+1)} = \arg \max_{(D_i, M_j) \in C(t)} f_{zoo}(\{D_i, M_j\}; Z(t))$$  \hspace{1cm} (7)

where $f_{zoo}((D, M); Z) := E_{\hat{\delta}_{acc}}[g_D(Z \cup \{D, M, \hat{\delta}_{acc}\}) - g_D(Z)]$, $\hat{\delta}_{acc} \sim S(\{D, M\}; \psi_{zoo})$ with parameter $\psi_{zoo}$ trained on $Z$, and the function $g_D$ measures the volume under the Pareto curve, also known as the Hypervolume Indicator [41], for the dataset $D$. We then train $M$ on $D$, and add it to the current model zoo $Z(t)$. For the full algorithm, please refer to Appendix A.

### 4 Experiments

In this section, we conduct extensive experimental validation against conventional NAS methods and commercially available AutoML platforms, to demonstrate the effectiveness of our proposed method.

#### 4.1 Experimental Setup

**Datasets**  
We collect 96 real-world image classification datasets from Kaggle*. Then we divide the datasets into two non-overlapping sets for 86 meta-training and 10 meta-test datasets. As some datasets contain relatively large number of classes than the other datasets, we adjust each dataset

---

*https://www.kaggle.com/*
to have up to 20 classes, yielding 140 and 10 datasets for meta-training and meta-test datasets, respectively (Please see Table 5 for detailed dataset configuration). For each dataset, we use randomly sampled 80/20% instances as a training and test set. To be specific, our 10 meta-test datasets include Colorectal Histology, Drawing, Dessert, Chinese Characters, Speed Limit Signs, Alien vs Predator, COVID-19, Gemstones, and Dog Breeds. We strictly verified that there is no dataset-, class-, and instance-level overlap between the meta-training and the meta-test datasets, while some datasets may contain semantically similar classes.

**Baseline Models** We consider MobileNetV3 [26] pretrained on ImageNet as our baseline neural network. We compare our method with conventional NAS methods, such as PC-DARTS [65] and DrNAS [10], weight-sharing approaches, such as FBNet [60] and Once-For-All [8], and data-driven meta-NAS approaches, e.g. MetaD2A [31]. All these NAS baseline approaches are based on MobileNetV3 pretrained on ImageNet, except for the conventional NAS methods that are only able to generate architectures. As such conventional NAS methods start from the scratch, we train them for sufficient amount of training epochs (10 times more training steps) for fair comparison. Please see Appendix B for further detailed descriptions of the experimental setup.

**Model-zoo Construction** We follow the OFA search space [8], which allows us to design resource-efficient architectures, and thus we obtain network candidates from the OFA space. We sample 100 networks candidates per meta-training dataset, and then train the network-dataset pairs, yielding 14,000 dataset-network pairs in our full model-zoo. We also construct smaller-sized efficient model-zoos from the full model-zoo (14,000) with our efficient algorithm described in Section 3.3. We use the full-sized model-zoo as our base model-zoo, unless otherwise we clearly mention the size of the used model-zoo. Detailed descriptions, e.g. training details and costs, are described in Appendix D.2.

### 4.2 Experimental Results

**Meta-test Results** We compare 50-epoch accuracy between TANS and the existing NAS methods on 10 novel real-world datasets. For a fair comparison, we train PC-DARTS [65] and DrNAS [10] for 10 times more epochs (500), as they only generate architectures without pretrained weights, so we train them for a sufficient amount of iterations. For FBNet and OFA (weight-sharing methods) and

<table>
<thead>
<tr>
<th>Target Dataset</th>
<th>Method</th>
<th># Epochs</th>
<th>FLOPs (M)</th>
<th>Params (M)</th>
<th>Search Time (GPU sec)</th>
<th>Training Time (GPU sec)</th>
<th>Speed Up</th>
<th>Accuracy (M) (±)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colorectal Histology Dataset (Easy)</td>
<td>MobileNetV3 [26]</td>
<td>50</td>
<td>132.94</td>
<td>4.00</td>
<td>257.78 ± 0.42</td>
<td>1.90 ± 0.08</td>
<td>94.20 ± 0.93</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PC-DARTS [65]</td>
<td>500</td>
<td>566.55</td>
<td>3.54</td>
<td>1100.37 ± 22.26</td>
<td>5721.15 ± 7.95</td>
<td>0.04 ± 0.03</td>
<td>79.22 ± 1.60</td>
</tr>
<tr>
<td></td>
<td>OFA [8]</td>
<td>50</td>
<td>623.43</td>
<td>4.12</td>
<td>1501.75 ± 0.42</td>
<td>6597.77 ± 4.62</td>
<td>0.04 ± 0.04</td>
<td>84.06 ± 0.97</td>
</tr>
<tr>
<td></td>
<td>MetaD2A [31]</td>
<td>50</td>
<td>512.67</td>
<td>6.74</td>
<td>2429.42 ± 0.45</td>
<td>7359.77 ± 0.68</td>
<td>0.88 ± 0.05</td>
<td>95.30 ± 1.15</td>
</tr>
<tr>
<td></td>
<td>TANS (Ours)</td>
<td>10</td>
<td>181.74</td>
<td>5.51</td>
<td>0.002 ± 0.00</td>
<td>50.17 ± 0.05</td>
<td>97.17 ± 1.20</td>
<td>96.28 ± 1.30</td>
</tr>
</tbody>
</table>

| Colorectal Histology Dataset (Hard) | MobileNetV3 [26] | 50       | 132.94    | 4.00       | 257.78 ± 0.42        | 1.90 ± 0.08            | 94.20 ± 0.93     |
|                                    | PC-DARTS [65]   | 500      | 566.55    | 3.54       | 1100.37 ± 22.26      | 5721.15 ± 7.95        | 0.04 ± 0.03   | 79.22 ± 1.60   |
|                                    | OFA [8]         | 50       | 623.43    | 4.12       | 1501.75 ± 0.42       | 6597.77 ± 4.62        | 0.04 ± 0.04   | 84.06 ± 0.97   |
|                                    | MetaD2A [31]    | 50       | 512.67    | 6.74       | 2429.42 ± 0.45       | 7359.77 ± 0.68        | 0.88 ± 0.05   | 95.30 ± 1.15   |
|                                    | TANS (Ours)     | 10       | 181.74    | 5.51       | 0.002 ± 0.00         | 50.17 ± 0.05          | 97.17 ± 1.20  | 96.28 ± 1.30   |

| Food Classification Dataset | MobileNetV3 [26] | 50       | 132.94    | 4.00       | 257.78 ± 0.42        | 1.90 ± 0.08            | 94.20 ± 0.93     |
|                            | PC-DARTS [65]   | 500      | 566.55    | 3.54       | 1100.37 ± 22.26      | 5721.15 ± 7.95        | 0.04 ± 0.03   | 79.22 ± 1.60   |
|                            | OFA [8]         | 50       | 623.43    | 4.12       | 1501.75 ± 0.42       | 6597.77 ± 4.62        | 0.04 ± 0.04   | 84.06 ± 0.97   |
|                            | MetaD2A [31]    | 50       | 512.67    | 6.74       | 2429.42 ± 0.45       | 7359.77 ± 0.68        | 0.88 ± 0.05   | 95.30 ± 1.15   |
|                            | TANS (Ours)     | 10       | 181.74    | 5.51       | 0.002 ± 0.00         | 50.17 ± 0.05          | 97.17 ± 1.20  | 96.28 ± 1.30   |

**Figure 4: Retrieved examples for meta-test datasets.** We visualize the dataset used for pretraining the retrieved model with the unseen query dataset. For more examples, see Figure 9 in Appendix.
The Number of Parameters

<table>
<thead>
<tr>
<th>Method</th>
<th>Averaged Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBNet-A [60]</td>
<td>93.00±1.95</td>
</tr>
<tr>
<td>OFA [8]</td>
<td>93.89±0.84</td>
</tr>
<tr>
<td>MetaD2A [31]</td>
<td>95.24±1.14</td>
</tr>
<tr>
<td>TANS w/o Topol.</td>
<td>95.24±0.21</td>
</tr>
<tr>
<td>TANS (Ours)</td>
<td>96.28±0.30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method (1/20 Model-Zoo)</th>
<th>Averaged Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TANS (Ours)</td>
<td>95.07%</td>
</tr>
<tr>
<td>TANS w/ Random Init.</td>
<td>74.89%</td>
</tr>
<tr>
<td>TANS w/ ImageNet</td>
<td>94.84%</td>
</tr>
<tr>
<td>TANS w/MetaD2A Arch.</td>
<td>95.38%</td>
</tr>
</tbody>
</table>

Figure 6: In-depth analysis of TANS: Performance comparison with constraints, such as (a) the number of parameters and (b) FLOPs. (c) Visualization for the cross-modal latent space using T-SNE and performance comparison depending on the distance (d) Ablation study on topology information. (e) Analysis on the architecture and pretrained knowledge.

MetaD2A (data-driven meta-NAS), which provide the searched architectures as well as pretrained weights from ImageNet-1K, we fine-tune them on the meta-test query datasets for 50 epochs.

As shown in Table 1, we observe that TANS outperforms all baselines, with incomparably smaller search time and relatively smaller training time. Conventional NAS approaches such as PC-DARTS and DrNAS repeatedly require large search time for every dataset, and thus are inefficient for this practical setting with real-world datasets. FBNet, OFA, and MetaD2A are much more efficient than general NAS methods since they search for subnets within a given supernet, but obtain suboptimal performances on unseen real-world datasets as they may have largely different distributions from the dataset the supernet is trained on. In contrast, our method achieves almost zero cost in search time, and reduced training time as it fine-tunes a network pretrained on a relevant dataset. In Figure 5, we show the test performance curves and observe that TANS often starts with a higher starting point, and converges faster to higher accuracy.

In Figure 4, we show example images from the query and training datasets that the retrieved models are pretrained on. In most cases, our method matches semantically similar datasets to the query dataset. Even for the semantically-dissimilar cases (right column), for which our model-zoo does not contain models pretrained on datasets similar to the query, our models still outperform all other base NAS models. As such, our model effectively retrieves not only task-relevant models, but also potentially best-fitted models even trained with dissimilar datasets, for the given query datasets. We provide detailed descriptions for all query and retrieval pairs in Figure 9 of Appendix.

We also compare with commercially available AutoML platforms, such as Microsoft Azure Custom Vision [1] and Google Vision Edge [2]. For this experiment, we evaluate on randomly chosen five datasets (out of ten), due to excessive training costs required on AutoML platforms. As shown in Figure 2, our method outperforms all commercial NAS/AutoML methods, with a significantly smaller total time cost. We provide more details and additional experiments, such as including real-world architectures, in Appendix E.

Analysis of the Architecture & Parameters To see that our method is effective in retrieving networks with both optimal architecture and relevant parameters, we conduct several ablation studies. We first report the results of base models that only search for the optimal architectures. Then we provide the results of the network retrieved using a variant of our method which does not use the topology (architecture) embedding, and only uses the functional embedding \( v_f \) (Tans w/o Topol). As shown in Figure 6 (d), TANS w/o Topol outperforms base NAS methods (except for MetaD2A) without considering the architectures, which shows that the performance improvement mostly comes from the knowledge transfer from the weights of the most relevant pretrained networks. However, the full TANS obtains about 1% higher performance over TANS w/o Topol., which shows the importance

Figure 5: Meta-test Accuracy Curves

(c) The Cross-Modal Latent Space

(d) Effectiveness of Topology

(e) Analysis on Arch. & Parameters

Figure 6: In-depth analysis of TANS: Performance comparison with constraints, such as (a) the number of parameters and (b) FLOPs. (c) Visualization for the cross-modal latent space using T-SNE and performance comparison depending on the distance (d) Ablation study on topology information. (e) Analysis on the architecture and pretrained knowledge.

MetaD2A (data-driven meta-NAS), which provide the searched architectures as well as pretrained weights from ImageNet-1K, we fine-tune them on the meta-test query datasets for 50 epochs.

As shown in Table 1, we observe that TANS outperforms all baselines, with incomparably smaller search time and relatively smaller training time. Conventional NAS approaches such as PC-DARTS and DrNAS repeatedly require large search time for every dataset, and thus are inefficient for this practical setting with real-world datasets. FBNet, OFA, and MetaD2A are much more efficient than general NAS methods since they search for subnets within a given supernet, but obtain suboptimal performances on unseen real-world datasets as they may have largely different distributions from the dataset the supernet is trained on. In contrast, our method achieves almost zero cost in search time, and reduced training time as it fine-tunes a network pretrained on a relevant dataset. In Figure 5, we show the test performance curves and observe that TANS often starts with a higher starting point, and converges faster to higher accuracy.

In Figure 4, we show example images from the query and training datasets that the retrieved models are pretrained on. In most cases, our method matches semantically similar datasets to the query dataset. Even for the semantically-dissimilar cases (right column), for which our model-zoo does not contain models pretrained on datasets similar to the query, our models still outperform all other base NAS models. As such, our model effectively retrieves not only task-relevant models, but also potentially best-fitted models even trained with dissimilar datasets, for the given query datasets. We provide detailed descriptions for all query and retrieval pairs in Figure 9 of Appendix.

We also compare with commercially available AutoML platforms, such as Microsoft Azure Custom Vision [1] and Google Vision Edge [2]. For this experiment, we evaluate on randomly chosen five datasets (out of ten), due to excessive training costs required on AutoML platforms. As shown in Figure 2, our method outperforms all commercial NAS/AutoML methods, with a significantly smaller total time cost. We provide more details and additional experiments, such as including real-world architectures, in Appendix E.

Analysis of the Architecture & Parameters To see that our method is effective in retrieving networks with both optimal architecture and relevant parameters, we conduct several ablation studies. We first report the results of base models that only search for the optimal architectures. Then we provide the results of the network retrieved using a variant of our method which does not use the topology (architecture) embedding, and only uses the functional embedding \( v_f \) (Tans w/o Topol). As shown in Figure 6 (d), TANS w/o Topol outperforms base NAS methods (except for MetaD2A) without considering the architectures, which shows that the performance improvement mostly comes from the knowledge transfer from the weights of the most relevant pretrained networks. However, the full TANS obtains about 1% higher performance over TANS w/o Topol., which shows the importance
We first visualize the meta-learned latent space in Figure 6 (c) with randomly sampled 1,400 models which makes the performance gap across models small. In sum, as the task (dataset) becomes more difficult, we can observe a higher correlation in the latent space between the distance and the rank.

We also show Spearman correlation scores on 5 meta-test datasets in Table 2. Measuring correlation with the distances is not directly compatible with weights and ImageNet pre-trained weights, using 1/20 sized model-zoo (700). We observe that they replace the contrastive loss with a ranking loss instead, but this will not be very meaningful. Hence, to effectively validate our latent space with correlation metric, we rather select clusters, such that 50 models around the query point and another 50 models around the farthest points, thus using a total of 100 models to report the correlation scores. In the Table 2, we show the correlation scores of these 100 models on the five unseen datasets. For Food dataset (reported as "hard" dataset in Table 1), the correlation scores are shown to be high. On the other hand, for Colorectal Histology dataset (reported as "easy" dataset), the correlation scores are low as any model can obtain good performance on it, compared to networks embedded the closest to the query dataset achieves higher performance on it, compared to networks embedded the farthest from it. For example, the accuracy of the closest network point for UCF-AI is 98.94% while the farthest network only achieves 91.53%.

We also show Spearman correlation scores on 5 meta-test datasets in Table 2. Measuring correlation with the distances is not directly compatible with our contrastive loss, since the negative examples (models that achieve low performance on the target dataset) are pushed away from the query point, without a meaningful ranking between the negative instances. To obtain a latent space where the negative examples are also well-ranked, we should replace the contrastive loss with a ranking loss instead, but this will not be very meaningful. Hence, to effectively validate our latent space with correlation metric, we rather select clusters, such that 50 models around the query point and another 50 models around the farthest points, thus using a total of 100 models to report the correlation scores. In the Table 2, we show the correlation scores of these 100 models on the five unseen datasets. For Food dataset (reported as "hard" dataset in Table 1), the correlation scores are shown to be high. On the other hand, for Colorectal Histology dataset (reported as "easy" dataset), the correlation scores are low as any model can obtain good performance on it, which makes the performance gap across models small. In sum, as the task (dataset) becomes more difficult, we can observe a higher correlation in the latent space between the distance and the rank.

**Figure 7:** In-depth analysis of TANS (2): (a) The effectiveness of meta-performance predictor. (b) Ablation study on retrieval performance. Additional (c) analysis and (d) the effectiveness of our performance predictor.
We propose a novel Task-Adaptive Neural Network Search framework (TANS), that instantly retrieves the best-fitted neural network for unseen data. The results show that the proposed TANS rapidly searches and trains a well-fitted network for unseen datasets with almost no architecture search cost and significantly fewer fine-tuning steps to reach the target performance, compared to other NAS methods. We discuss the limitation and the societal impact of our work in Appendix F.

### Meta-performance Predictor

The role of the proposed performance predictor is not only guiding the model and query encoder to learn the better latent space but also selecting the best model among retrieval candidates. To verify its effectiveness, we measure the performance gap between the top-1 retrieved model w/o the predictor and the model with the highest scores selected using the predictor among retrieved candidates. As shown in Figure 7 (a), there are 1.5% - 8% performance gains on the meta-test datasets. The top-1 retrieved model from the model zoo with TANS may not be always optimal for an unseen dataset, and our performance predictor remedies this issue by selecting the best-fitted model based on its estimation. We also examine ablation study for our performance predictor. Please note that we do not use ranking loss which does not rank the negative examples. Thus we use Mean Squared Error (MSE) scores. We retrieve the top 10 most relevant models for an unseen query datasets and then compute the MSE between the estimated scores and the actual ground truth accuracies. As shown in Figure 7 (c), we observe that removing either query or model embeddings degrades performance compared to the predictor taking both embeddings. It is natural that, with only model or query information, it is difficult to correctly estimate the accuracy since the predictor fails to recognize what or where to learn. Also, we report the MSE between the predicted performance using the predictor and the ground truth performance of each model for the entire set of pretrained models from a smaller model zoo in Figure 7 (d). Although the performance predictor achieves slightly higher MSE scores for this experiment compared to the MSE obtained on the top-10 retrieved models (which are the most important), the MSE scores are still meaningfully low, which implies that our performance model successfully works even toward the entire model-zoo.

### Retrieval Performance

We also verify whether our model successfully retrieves the same paired models when the correspondent meta-train datasets are given (we use unseen instances that are not used when training the encoders.) For the evaluation metric, we use recall at k (R@k) which indicates the percentage of the correct models retrieved among the top-k candidates for the unseen query instances, where k is set to 1, 5, and 10. Also, we report the mean and median ranking of the correct network among all networks for the unseen query. In Figure 7 (b), the largest parameter selection strategy shows poor performances on the retrieval problem, suggesting that simply selecting the largest network is not a suitable choice for real-world tasks. In addition, compared to cosine similarity learning, the proposed meta-contrastive learning allows the model to learn significantly improved discriminative latent space for cross-modal retrieval. Moreover, without our performance predictor, we observe that TANS achieves slightly lower performance, while it is significantly degenerated when training without functional embeddings.

### Analysis of Model-Zoo Construction

Unlike most existing NAS methods, which repeatedly search the optimal architectures per dataset from their search spaces, TANS do not need to perform such repetitive search procedure, once the model-zoo is built beforehand. We are able to adaptively retrieve the relevant pretrained models for any number of datasets from our model-zoo, with almost zero search costs. Formally, TANS reduces the time complexities of both search cost and pre-training cost from $O(N)$ to $O(1)$, where $N$ is the number of datasets, as shown in Figure 8 (Top). Furthermore, a model zoo constructed using our efficient construction algorithm, introduced in Section 3.3, yields models with higher performance on average, compared to the random sampling strategy when the size of the model-zoo is the same as shown in Figure 8 (Bottom).

### Conclusion

We propose a novel Task-Adaptive Neural Network Search framework (TANS), that instantly retrieves a relevant pre-trained network for an unseen dataset, based on the cross-modal retrieval of dataset-network pairs. We train this retrieval model via amortized meta-learning of the cross-modal latent space with contrastive learning, to maximize the similarity between the positive dataset-network pairs, and minimize the similarity between the negative pairs. We train our framework on a model zoo consisting of diverse pretrained networks, and validate its performance on ten unseen datasets. The results show that the proposed TANS rapidly searches and trains a well-fitted network for unseen datasets with almost no architecture search cost and significantly fewer fine-tuning steps to reach the target performance, compared to other NAS methods. We discuss the limitation and the societal impact of our work in Appendix F.
Acknowledgement  This work was supported by AITRICS and by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST))

References


Algorithm 1: Model Zoo Construction

Input : $D, M$: collection of datasets and models, respectively, $Z_{(t)} \subset D \times M \times \mathbb{R}_{[0,1]}$: set of $N_{init}$ initial tuples of (dataset, model, test accuracy) $t \leftarrow 0$;

while termination condition is not met do
  if $t$ is divisible by $N_{train}$, then
    Train accuracy predictor parameters $\psi_{zoo}$ on data $Z_{(t)}$;
    $C_{(t)} \leftarrow$ Choose a subset of candidate pairs from $D \times M$ not present in $Z_{(t)}$;
    $(D, M) \leftarrow \max_{(D,M) \in C_{(t)}} f_{zoo}(C; Z_{(t)});
    \alpha^* \leftarrow$ Evaluate the actual accuracy of $(D, M)$ by training $M$ on $D$;
    $Z_{(t+1)} \leftarrow Z_{(t)} \cup (D, M, \alpha^*)$;
  $t \leftarrow t + 1$;

Output : Efficiently constructed model zoo $Z_{(t)}$.

The algorithm that is used to efficiently construct the model-zoo is described in Algorithm 1. The score function $f_{zoo}(D, M; Z)$, which measures how much adding a pair $(D, M)$ will improve upon the model zoo $Z$, is defined as

$$f_{zoo}(D; M; Z) := \text{E}_{\tilde{h}_{acc} \sim S((D, M); \psi_{zoo})}[g_D(Z \cup (D, M, \tilde{h}_{acc})) - g_D(Z)]$$

where $S$ indicates the accuracy predictor, and $g_D$ is the normalized volume under the pareto-dominated pairs for dataset $D$:

$$g_D \left( \left\{ (D^{(i)}, M^{(i)}, \tilde{h}_{acc}^{(i)}) \right\}_{i=1}^n \right) := \text{Hypervolume} \left( \left\{ (\tilde{h}_{latency}(M^{(i)}), \tilde{h}_{parameters}(M^{(i)})) \left| D^{(i)} = D \right. \right\} \right)$$  

where $\tilde{h}_{latency}(M)$ and $\tilde{h}_{parameters}(M)$ indicates the normalized latency and the normalized number of parameters of the model $M$, respectively. The latency and parameters are normalized so that the maximum value across all models becomes 1.0 and the minimum value becomes 0.0. The hypervolume can be computed efficiently with the PyGMO library [5].

The accuracy predictor used in the model-zoo construction is very similar to the structure of the performance predictor described in Section A.4, but we used a functional embedding obtained from the model pretrained on Imagenet-1K, instead of a functional embedding obtained from a model already trained on the target dataset, since training the model on the target dataset just to obtain the functional embedding would defeat the purpose of this algorithm. Also, to incorporate uncertainty about the accuracy predictions, we use 10 samples from the accuracy predictor with MC dropout to evaluate the expectation in (8). The dropout probability is set to 0.5.

A.2 Query Encoder

Our query encoder takes sampled instances (e.g. 10 unseen random images per class) as an input from the query dataset. We use image embeddings from ResNet18 [23] pretrained on ImageNet 1K [13], whose dimensions...
are 512 (except for the last classification layer), rather than using raw images, simply to reduce computation costs. We then use a linear layer with 512 dimensions, followed by Mean pooling and $L_2$ normalization, which outputs encoded vectors with 128 dimensions. As we use Deep set [69], we tried performing Sum pooling, instead of Mean pooling, however, we observe that taking the average on instances shows better R@$k$ scores for the correct pair retrieval, and thus we use Mean pooling when encoding query samples.

### A.3 Model Encoder

Our model encoder takes both OFA flat topology [8] and functional embedding as an input. For the flat topology, it contains information such as kernel size, width expansion ratio, and depth, in a 45-dimensional vector. In addition, the functional embedding, which bypasses the need for direct parameter encoding, represents models’ learned knowledge in a 1536-dimensional vector. We first concatenate both vectors and normalize the vector. Then, we learn a projection layer, which is a 1581-length fully-connected layer, followed by $L_2$ norm operation, which outputs encoded vectors with 128 dimensions.

### A.4 Meta-Surrogate Performance Predictor

Our performance predictor takes both query and model embeddings simultaneously. Both embedding vectors are 128-dimensional vectors. We first concatenate the embeddings into 256-dimensional vectors and then forward them through a 256-length fully connected layer. We then produce a single continuous value for a predicted accuracy. We perform a sigmoid operation on the values to range the values into a scale from 0.0 to 1.0.

### B Training Details

There are two steps of training required for our Task-Adaptive Network Search (TANS): 1) training the cross-modal latent space and 2) fine-tuning the retrieved model on an unseen meta-test dataset.

#### B.1 Learning the Cross-modal Latent Space

For the model-zoo encoding, we set the batch size to 140 as we have 140 different datasets. Since, for each dataset, we randomly choose one model among 100 models from each dataset. Then we minimize the contrastive loss on the 140 samples. Although we train our encoders over a large number of dataset-network pairs (14,000 models), the entire training time takes less than two hours based on NVIDIA’s RTX 2080 TI GPU. We initialize our model weights with the same value across all encoders and experiments, rather than differently initializing the encoders for every experimental trial. We use the Adam optimizer (We use the learning rate of $1 \times 10^{-2}$).

#### B.2 Fine-tuning on Meta-test Datasets

For the fine-tuning phase, we set all settings, such as hyper-parameters, learning rate, optimizer, etc., exactly the same across all baseline models and our method, and the differences are clearly mentioned in this section otherwise. We use the SGD optimizer with an initial learning rate (1e-2), weight decay (4e-5), and momentum (0.9). Also, we use the Cosine Annealing learning rate scheduler. We train the models with 224 sized images (after resizing) and we set batch-size to 32, except PC-DARTS which has memory issues with 224 sized images (for PC-DARTS, we set to 12 as the batch-size), and DrNAS which we train with 32 sized images due to heavy training time costs. We train all models for 50 epochs and we show that our model converges faster than all baseline models.

#### B.3 Constructing the Model-Zoo

For the model-zoo consisting of 14,000 random pairs used in the main experiment, we fine-tune the ImageNet1K pretrained OFA models on the dataset for 625 epochs, following the progressive shrinking method described in [8]. We then choose 100 random OFA architectures for each dataset and evaluate their test accuracies on the test split.

For the efficiently constructed model zoo experiment, we use the algorithm described in Section 3.3 and further elaborated in Section A.1, using the 14,000-pair model zoo as the search space. For the initial samples, we use $N_{init} = 750$, where 5-6 samples are taken from each dataset. The accuracy predictor is retrained from scratch every 64 iterations until the validation accuracy no longer improves for 5 epochs.

### C Proof for Uniqueness of the Query and Model Encoding Functions

In this section, we show that the proposed query and model encoders can represent the injective function on the input query $D \in \mathcal{D}$ and model $M \in \mathcal{M}$, respectively.
We first show that the topological encoding function is injective. Assume that a computational graph for a neural network with both topological embedding and functional embedding has arbitrary input signal $v_i$ and operation $o_i$, with the computational graph represented as the DAG structure. Then, our goal here is to make a query encoder that uniquely maps different queries $D_1, D_2$ into two distinct embeddings $q_1, q_2$.

Each dataset $D$ consists of $n$ data instances: $D = \{X_1, X_2, \ldots, X_n\}$, where $n$ is smaller than the number of elements in $\mathbb{N}$. To encode each query dataset $D$ into a vector space, as described in query encoder paragraph of section 3.2, we first transform each instance $X_i$ into the representation space with a continuous function $\rho$, and then aggregate all set elements, which is adopted from Zaheer et al. [69]. In other words, a query encoder can be defined as follows: $q = \sum_{X_i \in D} \rho(X_i)$.

We assume that $Q$ is a finite set, and each $D = \{X_1, X_2, \ldots, X_n\} \in Q$ is also a finite set with $|D| = n$ elements. Therefore, a set of data instances $X$ is countable, since the product of two nature numbers (i.e., $|Q|\times n$) is a natural number. For this reason, there can be a unique mapping $\pi$ from the element $X$ to the nature number in $\mathbb{N}$. If we let $\rho(X) = 4^{-\pi(X)}$, then the form of a query encoder $\sum_{X_i \in D} \rho(X_i)$ constitutes a unique mapping for every set $D \in Q$ (see Zaheer et al. [69], Wagstaff et al. [57] for details). In other words, the output of the query encoder is unique for each input dataset $D$ that consists of $n$ data instances.

Thanks to the universal approximation theorem [25, 24], we can construct a mapping function $\rho$ using multi-layer perceptrons (MLPs).

**Proposition 1 (Injectiveness on Query Encoding).** Assume $Q$ and $D$ are finite sets. A query encoder $E_Q : Q \to R^d$ can injectively map two different queries $D_1, D_2$ into distinct embeddings $q_1, q_2$, where $D \in Q$ and $q \in R^d$.

**Proof.** A query encoder $E_Q$ maps a query dataset $D \in Q$ to a vector $q \in R^d$ as follows: $E_Q : D \mapsto q$, where $Q$ is a set of queries, which contains a set of data instances $X$ for constructing a dataset $D = \{X_1, X_2, \ldots, X_n\}$. Then, our goal here is to make a query encoder that uniquely maps two different queries $D_1, D_2$ into two distinct embeddings $q_1, q_2$.

We assume that $Q$ and $D$ are finite sets, and each $D = \{X_1, X_2, \ldots, X_n\} \in Q$ is also a finite set with $|D| = n$ elements. Therefore, a set of data instances $X$ is countable, since the product of two nature numbers (i.e., $|Q|\times n$) is a natural number. For this reason, there can be a unique mapping $\pi$ from the element $X$ to the nature number in $\mathbb{N}$. If we let $\rho(X) = 4^{-\pi(X)}$, then the form of a query encoder $\sum_{X_i \in D} \rho(X_i)$ constitutes a unique mapping for every set $D \in Q$ (see Zaheer et al. [69], Wagstaff et al. [57] for details). In other words, the output of the query encoder is unique for each input dataset $D$ that consists of $n$ data instances.

While we can obtain the distinct embedding of each neural network with the topological encoding function alone, we also consider the injectiveness of the functional encoding in the following. To consider the functional embedding, we first model a neural architecture as its computational graph, which can be further denoted as a directed acyclic graph (DAG). Using this computational graph scheme, a functional model encoder $E_{M_f}$ maps an architecture (computational graph) $M \in \mathcal{M}$ into a vector $v_f$ as follows: $E_{M_f} : M \mapsto v_f$. Then, our goal here is to make the functional encoder $E_{M_f}$, that uniquely maps two different neural architectures $M_1, M_2$ into two different embeddings $v_{f1}, v_{f2}$, with the computational graph represented as the DAG structure.

**Proposition 2 (Injectiveness on Model Encoding).** Assume $\mathcal{M}$ is a countable set. A model encoder $E_M : \mathcal{M} \to R^d$ can injectively map two different architectures $M_1, M_2$ into distinct embeddings $m_1, m_2$, where $M \in \mathcal{M}$ and $m \in R^d$.

**Proof.** As described in the model encoder paragraph of Section 3.2, we represent each neural network $M \in \mathcal{M}$ with both topological embedding $v_t$ and functional embedding $v_f$. Thus, if one of two embeddings can uniquely represent each neural network, then the injectiveness on model encoding $E_M : \mathcal{M} \to R^d$ is satisfied.

We first show that the topological encoding function $E_{M_f} : M \mapsto v_t$ can uniquely represent each architecture $M$ in the embedding space. As described in the Model Encoder part of section A, we use a 45-dimensional vector that contains topological information, such as the number of layers, channel expansion ratios, and kernel sizes (See Cai et al. [8] for details), for the topological encoding. Also, each topological information uniquely defines each neural architecture. Therefore, the embedding $v_t$ from the topological encoding function $E_{M_f}$ is unique on each neural network $M$.

While we can obtain the distinct embedding of each neural network with the topological encoding function alone, we also consider the injectiveness of the functional encoding in the following. To consider the functional encoding, we first model a neural architecture as its computational graph, which can be further denoted as a directed acyclic graph (DAG). Using this computational graph scheme, a functional model encoder $E_{M_f}$ maps an architecture (computational graph) $M \in \mathcal{M}$ into a vector $v_f$ as follows: $E_{M_f} : M \mapsto v_f$. Then, our goal here is to make the functional encoder $E_{M_f}$, that uniquely maps two different neural architectures $M_1, M_2$ into two different embeddings $v_{f1}, v_{f2}$, with the computational graph represented as the DAG structure.

Assume that a computational graph for a neural network $M$ has $n$ nodes. Then, each node $v_i$ on the graph has its corresponding operation $o_i$, which transforms incoming features for the node $v_i$ to an output representation $C_i$. In other words, $C_i$ indicates the output of the composition of all operations along the path from $v_1$ to $v_i$.

Particularly, in our model encoder case, we have an arbitrary input signal $\mathbf{x}$ that is the fixed Gaussian noise, where we insert this fixed input into the starting node $v_1$ (See Model Encoder paragraph of section 3.2 for details). Also, for the simplicity of the notation, we set $C_0 = \mathbf{x}$ that is the output of the virtual node $v_0$ and the incoming representation of the starting node of the computational graph. Then, the output representation for the node $v_i$ is formally defined as follows: $C_i(\mathbf{x}) = o_i(\{C_j(\mathbf{x}) : v_j \to v_i\})$, where $\{C_j(\mathbf{x}) : v_j \to v_i\}$ denotes a multiset for the output representation of $v_i$’s predecessors, and the operation $o_i$ transforms the incoming representations over the multiset into the output representation. Note that, to consider the multiplicity of the nodes on a graph, we use a multiset scheme, rather than a set [64].

Following the proof of Theorem 2 in Zhang et al. [72], we rewrite the $C_i(\mathbf{x}) = o_i(\{C_j(\mathbf{x}) : v_j \to v_i\})$ as follows: $C_i(\mathbf{x}) = o_i(\{C_j(\mathbf{x}) : v_j \to v_i\})$, where $\omega$ is an injective function over two inputs $o_i$ and $\{C_j(\mathbf{x}) : v_j \to v_i\}$. Then, $C_i$ can uniquely embed the output representation of node $v_i$, and this is an injective (Please refer to the proof of Theorem 2 in Zhang et al. [72] for details). Thus, the output of the computational
We present all query-retrieval pairs on meta-test datasets. Each row includes two pairs of query dataset (left) and the retrieved dataset (right). Please see detailed explanations of the pairs in Section D.3 and Table 5.

We show that the model representation with both topological encoding and functional encoding schemes, although only one encoding function can injectively represent the entire neural network. We further concatenate and transform two output representations with a function \( g \), to obtain the final model representation:

\[
\mathbf{m} = g(\mathbf{v}_t, \mathbf{v}_f)
\]

The representation \( \mathbf{m} \) is also unique on each neural network \( M \) with an injective function \( g \).

To sum up, we show the injectiveness of the model representation with both topological encoding and functional encoding schemes, although only one encoding function can injectively represent the entire neural network.

We further concatenate and transform two output representations with a function \( g \), to obtain the final model representation:

\[
\mathbf{m} = g(\mathbf{v}_t, \mathbf{v}_f)
\]

The representation \( \mathbf{m} \) is also unique on each neural network \( M \) with an injective function \( g \).

Similar to the universal approximation theorem \([25, 24]\), we might construct an injective mapping function \( g \) and \( \omega \) with learnable parameters on it.

### D Experimental Setup

#### D.1 Architecture Space

Before constructing a model zoo that contains a large number of dataset-architecture pairs, we first need to define an architecture search space on it to handle all architectures in a consistent manner. To easily obtain the task-adaptive parameters for the given task with consideration of various factors, such as a number of layers, kernel sizes, and width expansion ratios, we use the supernet-based OFA architecture space \([8]\), which the same as the well-known MobileNetV3 space \([26]\). Each neural architecture in the search space consists of a stack of 20 mobile-block convs (MBconvs), where the number of units is 5, and the number of layers on each unit ranges across \( \{2, 3, 4\} \). Moreover, for each layer, we select the kernel size is from \( \{3, 5, 7\} \), and the width-depth ratio from \( \{3, 4, 6\} \). This strategy allows us to generate around \( 10^{19} \) neural architecture candidates in theory.
D.2 Model Zoo

To construct a model zoo consisting of a large number of dataset-architecture pairs, we collect 89 real-world datasets for image classification from Kaggle\footnote{https://www.kaggle.com/} and obtain 100 random architectures per dataset from the OFA space.Specifically, we first divide the collected datasets into two non-overlapping sets for meta-training and meta-testing. If the dataset has more than 20 classes, then we randomly split it into multiple datasets such that a dataset can consist of up to 20 classes. For meta-testing, we randomly selected only one of the splits for each original dataset for diversity. This process yields 140 datasets for meta-training and 10 datasets for meta-testing. To generate a validation set for each dataset, we randomly sample 20\% of data instances from each dataset and use the sampled instances for the validation, while using the remaining 80\% as the training instances.

For statistics, the number of classes ranges from 2 to 20 with a median of 16, and the number of instances for each dataset ranges from 8 to 158K with a mean of 2,847. We then construct the model-zoo by fine-tuning 100 random OFA architectures on training instances of each dataset and obtaining their performances on its respective validation instances, which yields 14K (dataset, architecture, accuracy) tuples in total. We use this database throughout this paper.

D.3 Dataset Details

In Table 5, we provide information of all datasets that we utilize for model-zoo construction and meta-test experiments, including the dataset name, a brief description of the dataset, the number of splits for train and test sets, and corresponding Kaggle URL. Please refer to the table if you look into a certain dataset more closely. Particularly, we further provide an explanation of all query and retrieval pairs in Figure 9. Beginning from the left column on the first row, we present pairs of Colorectal Histology (query) & Breast Histopathology (retrieval) and Real or Drawing (query) & Tiny Images (retrieval). In the second row, we show pairs of Dessert (query) & Food Kinds (retrieval) and Chinese Characters (query) & Vehicles (retrieval). In the third row, there are pairs of Speed Limit Signs (query) & Traffic Signs (retrieval) and Alien vs Predator (query) & Grid Anomaly (retrieval). In the fourth row, we illustrate pairs of COVID-19 (query) & Chest X-Rays (retrieval) and Gemstones (query) & Stanford Dogs (retrieval). In the last row, we present pairs of Dog Breeds (query) & Stanford Dogs (retrieval) and Honeybee Pollen (query) & Breast Cancer Tissues (retrieval). Please see Table 5 for the detailed information for each dataset.

D.4 Baseline NAS Methods

Here we describe the baselines we use in the experiments in the main document. We compare the performance of the models retrieved with our method against pretrained neural networks as well as the ones searched by several efficient NAS methods that are closely related to ours:

1) MobileNetV3 \[26\] MobileNetV3 is a representative resource-efficient neural architecture tuned considering mobile phone environments. In our experiments, MobileNetV3 is pretrained on ImageNet-1K, which is fine-tuned for 50 epochs on each meta-testing task.

2) PC-DARTS \[65\], a differentiable NAS method based on a weight sharing scheme that reduces search time efficiently and especially improves memory usage, search time, performance compared to DARTS \[35\] by designing partial channel sampling and edge normalization. We search for architectures for each meta-testing task by following the official code at https://github.com/yuhuixu1993/PC-DARTS.

3) DrNAS \[10\], a differentiable NAS method that handles NAS as a distribution problem, modeled by Dirichlet distribution. We use the official code at https://github.com/xiangning-chen/DrNAS.

4) OFA \[8\], a NAS method that provides a subnet sampled from a larger network (supernet) pretrained on ImageNet-1K, which alleviates the performance degeneration of prior supernet-based methods. We use the code at https://github.com/mit-han-lab/once-for-all.

5) MetaD2A \[31\], a meta-NAS model that rapidly generates data-dependent architecture for a given task that is meta-learned on subsets of ImageNet-1K. From the ImageNet-1K dataset and architectures of OFA search space, we randomly use 3296 and 14,000 meta-training tasks for the generator and predictor, respectively as a source database.

6) FBNet \[61\], a collection of convolutional models obtained via Differentiable Neural Architecture Search. We use FBNet-A pretrained on ImageNet-1K and fine-tune it on each meta-testing task for 50 epochs.

We use the same hyper-parameters for all baselines for a fair comparison. We fine-tune the architecture for 50 epochs on each meta-testing task. The SGD optimizer is used with a learning rate of 0.01, the momentum of 0.9, and 4e-5 weight decay. The image size is 224×224 and the batch size is 32.
Table 3: Performance Comparison on 5 Unseen Real-world Datasets

<table>
<thead>
<tr>
<th>Target Dataset</th>
<th>Method</th>
<th>Params (M)</th>
<th>Search Time (GPU sec)</th>
<th>Training Time (GPU sec)</th>
<th>Speed Up</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MobileNetV3 [26]</td>
<td>4.00</td>
<td>178.45 ± 0.16</td>
<td>1.00 ± 0.07</td>
<td>96.36 ± 1.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PC-DARTS [64] 500 Epochs</td>
<td>3.45</td>
<td>943.17 ± 1.44</td>
<td>4255.74 ± 0.06</td>
<td>0.03 ± 1.24</td>
<td></td>
</tr>
<tr>
<td>Averaged</td>
<td>DrNAS [10] 500 Epochs</td>
<td>4.12</td>
<td>873.44 ± 20.78</td>
<td>2445.22 ± 20.44</td>
<td>0.05 ± 1.24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OFA [8]</td>
<td>7.30</td>
<td>216.40 ± 27.92</td>
<td>1.00 ± 0.02</td>
<td>96.15 ± 2.46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MetaD2A [31]</td>
<td>6.15</td>
<td>121.90 ± 0.00</td>
<td>62.33 ± 0.33</td>
<td>0.63 ± 1.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TANS (Ours) w/ OFA-Based Model-Zoo</td>
<td>5.50</td>
<td>0.002 ± 0.00</td>
<td>121.18 ± 0.31</td>
<td>1.47 ×</td>
<td>97.79 ± 0.28</td>
</tr>
<tr>
<td></td>
<td>TANS (Ours) w/ Real-world Model-Zoo</td>
<td>5.43</td>
<td>0.001 ± 0.00</td>
<td>115.06 ± 14.82</td>
<td>1.55 ×</td>
<td>98.59 ± 2.39</td>
</tr>
</tbody>
</table>

E Additional Experiments & Analysis

E.1 Experiment on Real-world Networks

While the proposed TANS shows outstanding performances in a number of neural network search tasks with the manicured architecture search space described in Section D.1, it could be more beneficial if we search the best-fitted model on a given query dataset from the pretrained networks with real-world neural architectures (e.g. ResNet) trained on various datasets. For this even more realistic scenario for Neural Network Search (NNS), we construct our model-zoo including ten real-world architectures, such as ResNet18 [23], ShuffleNet V2 [39], MobileNet v2 [45], SqueezeNet [28], GoogLeNet [50], ResNetXt [62], AlexNet [29], MNASNet [53], EfficientNet-B0 [51], and LambdaResNet [4].

Experimental Setup To construct the new real-world model-zoo, we first meta-train the real-world architectures and merge the new dataset-network pairs (1,400 pairs) with the random subset of the previous model-zoo (about 5,000 OFA-based models), yielding about 6,500 models in the new model-zoo. The way of training is the same as the experiment introduced in the main document (Section 4.1) except that we only use functional embeddings, while topology information is not used when learning the cross-modal latent space (we exclude the topology information since encoding the topologies of real networks across various search spaces into a single uniform format is too complicated.) Including the real-world architectures, we first verify the retrieval performance of our model on the meta-train datasets, and our TANS achieves 90 for the R@1, 100 for the R@5 scores. The way of evaluating on the meta-test dataset is also the same as the experiments that we conducted in the main document (10 real-world meta-test datasets), except that we conduct experiments only on five datasets out of the ten datasets used in the experiments of the main document, due to the heavy training costs required for meta-testing. The selected datasets are Speed Limit Signs, Honey-bee Pollen, Alien-vs-Predator, Chinese Characters, and COVID-19 datasets (for detailed information for each dataset, please see Table D.3.)

Experimental Results In Table 3, our methods, both with OFA and the real-world architectures, outperform all baseline models, including MobileNetV3 (about 1.0% to 1.5% higher), PC-DARTS (about 17.5% to 18.0% higher), DrNAS (about 14.5% to 15.0% higher), FBNNet (about 1.5% to 2.0% higher), OFA (about 1.5% to 2.0% higher), and MetaD2A (about 0.5% to 1.0% higher). We observe that collecting more lightweight real-world neural network and dataset pairs (TANS w/ Real-world Model-Zoo) will allow our model to retrieve computationally efficient pretrained networks in a task-adaptive manner. Such data-driven nature is another advantage of our method since we can easily increase the performance of the model by collecting more pretrained networks that are readily available in many public databases.

E.2 Additional Performance Comparison with NAS Methods

In the experiment introduced in the main document (Table 1), we train DrNAS and PC-DARTS, which only generate architectures without pretrained weights, for 10 times more iterations (500 epochs) for a fair comparison (while the other methods, which share ImageNet pretrained knowledge, are trained for 50 epochs). In this experiment, rather than training for 500 epochs, we pretrain networks obtained by DrNAS and PC-DARTS on “ImageNet” and then fine-tune on two meta-test datasets (Colorectal Histology & Food Classification Datasets). As shown in Table 4, although pretraining on ImageNet improves their results, our methods, including TANS with 1/10 sized model-zoo (1400), still outperforms all baselines, which shows that retrieving and utilizing pretrained weights of relevant tasks is more effective than using ImageNet pre-trained weights.
E.3 Synergistic Effect of TANS and State-of-the-Art NAS Methods

Not only the real-world architectures but also any existing NAS methods can be successfully integrated with our retrieval framework by simply adding searched networks into our model-zoo. We demonstrate such synergistic effect of TANS and NAS methods in Figure 6 (e) of the main document. Constructing the model-zoo with neural architectures generated by MetaD2A, which is a state-of-the-art NAS method, improves our performance compared to the previous model-zoo that are simply sampled from the OFA search space. Considering that NAS approaches have been actively studied [31, 52, 7, 51, 49, 15] and pretrained models are often shared via open-source, we believe that the TANS framework has powerful potential to continuously improve its performance by absorbing such new models into the model-zoo.

F Discussion

F.1 Societal Impacts

Our framework, TANS, has the following beneficial societal impacts: (1) enhanced accessibility, (2) preservation of data privacy, and (3) the reduction of reproducing efforts.

**Enhanced accessibility** Since our Task-Adaptive Neural Network Search (TANS) framework allows anyone to instantly retrieve a full neural network that works well on the given task, by providing only a small set of data samples, it can greatly enhance the accessibility of AI to users with little knowledge and backgrounds. Moreover, it does not require large computational resources, unlike existing NAS or AutoML frameworks, which further helps with its accessibility. Finally, to allow everyone to benefit from our task-adaptive neural network search framework, we will publicly release our model-zoo, which currently contains more than 15K models, and open-source it. Then, anyone will be able to freely retrieve/update any models from our model-zoo.

**Preservation of data privacy** Our framework requires only a small set of sampled data instances to retrieve the task-adaptive neural network, unlike existing NAS/AutoML methods that require a large number of data instances to search optimal architectures for the target datasets. Thus, the data privacy is largely improved, and we can further allow the set encoding to take place on the client-side, rather than at the server. This will result in enhanced data privacy, as none of the raw data samples need to be submitted to the system.

**Reduction of reproducing efforts** Many ML researchers and engineers are wasting their time and labors, as well as the computational and monetary resources in reproducing existing models and fine-tuning them. TANS, since it instantly retrieves a task-relevant model from a model zoo that contains a large number of state-of-the-art networks pretrained on diverse real-world datasets, the users need not redesign networks or retrain them at excessive costs. Since we plan to populate the model zoo with more pretrained networks, the coverage of the dataset and architectures will become even broader as time goes on. Since training deep learning models often requires extremely large computing cost, which is costly in terms of energy consumption, and results in high carbon emissions, our method is also environment-friendly.

F.2 Limitations

As a prerequisite condition, our method must have a model-zoo which contains pretrained models that can cover diverse tasks and perform well on each given task. There exists a chance that TANS could be affected by biased initialization if the meta-training pool contains biased pretrained models. To prevent this issue, we can use existing techniques that ensure fairness when constructing a model-zoo, which identify and discard inappropriate datasets or models. There have been various studies for alleviating unjustified bias in machine learning systems. Fairness can be classified into individual fairness, treating similar users similarly [16, 68], and group fairness, measuring the statistical parity between subgroups, such as race or gender [70, 36, 22]. Optimizing fair metrics during training is achieved by regularizing the covariance between sensitive attributes and model predictions [59] and minimizing an adversarial ability to estimate sensitive attributes from model predictions [71]. At evaluation times, [3, 12] improves the generalizability for a fair classifier via two-player games. All these methods can be adopted when building our model-zoo.
Table 5: **Dataset Details** Detailed information, such as dataset name, description, and download link, about all datasets that we utilize are described (Due to the space limit, we provide hyperlinks to the webpage for the datasets, rather than printing the full website links.)

<table>
<thead>
<tr>
<th>No.</th>
<th>Dataset Name</th>
<th>Brief Description</th>
<th>Instances</th>
<th>Cls.</th>
<th>Splits</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Store Items</td>
<td>Classify store item images by color</td>
<td>4984 / 624</td>
<td>12</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Big Cats</td>
<td>Classify big cats by species</td>
<td>2875 / 360</td>
<td>4</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Deepfake Detection</td>
<td>Deepfake detection</td>
<td>12000 / 1500</td>
<td>2</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Food Kinds</td>
<td>Classify kinds of food</td>
<td>10580 / 1322</td>
<td>11</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Hair Color</td>
<td>Classify people by hair color</td>
<td>2560 / 320</td>
<td>4</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Apparels</td>
<td>Classify apparel images by kind and color</td>
<td>9091 / 1137</td>
<td>24</td>
<td>2 Link</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Manual Alphabet</td>
<td>Classify manual alphabet letters</td>
<td>69600 / 8700</td>
<td>29</td>
<td>2 Link</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Artworks</td>
<td>Classify artworks by artist</td>
<td>6997 / 877</td>
<td>51</td>
<td>3 Link</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Blood Cells</td>
<td>Identify blood cell types</td>
<td>9954 / 1244</td>
<td>4</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Breast Cancer Tissues</td>
<td>Identify breast cancer with microscope images</td>
<td>6323 / 789</td>
<td>8</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Breast Histopathology</td>
<td>Identify breast cancer with sample images</td>
<td>222018 / 27753</td>
<td>2</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Aerial Cactus</td>
<td>Identify cacti in aerial photos</td>
<td>17199 / 2150</td>
<td>2</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Car Models</td>
<td>Classify cars by model</td>
<td>3229 / 405</td>
<td>45</td>
<td>3 Link</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Cassava Leaf Disease</td>
<td>Identify type of leaf disease</td>
<td>17115 / 2141</td>
<td>5</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Celebrity Images</td>
<td>Classify celebrity images by attractiveness</td>
<td>161985 / 20248</td>
<td>2</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Chess Pieces</td>
<td>Identify chess pieces</td>
<td>437 / 54</td>
<td>6</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Russian Handwritten Letters</td>
<td>Classify Russian handwritten letters</td>
<td>11350 / 1419</td>
<td>33</td>
<td>2 Link</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>CT Images</td>
<td>Identify intracranial hemorrhage in CT scans</td>
<td>4255 / 532</td>
<td>2</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Corals</td>
<td>Identify types of coral</td>
<td>489 / 62</td>
<td>14</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Cracks</td>
<td>Detect cracks in pavements and walls</td>
<td>13570 / 1697</td>
<td>2</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Cactus Identification</td>
<td>Identify cactus in images</td>
<td>17199 / 2150</td>
<td>2</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Animals</td>
<td>Classify animal pictures by species</td>
<td>320 / 32</td>
<td>16</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Blink</td>
<td>Identify which eye is closed</td>
<td>3874 / 484</td>
<td>5</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Dogs</td>
<td>Classify breeds of dogs</td>
<td>12558 / 1571</td>
<td>120</td>
<td>6 Link</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Furniture</td>
<td>Identify type of furniture</td>
<td>5186 / 648</td>
<td>5</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Forest Fire</td>
<td>Detect whether there is a fire in forest images</td>
<td>794 / 100</td>
<td>3</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Devanagari Characters</td>
<td>Identify Devanagari characters</td>
<td>73580 / 9197</td>
<td>46</td>
<td>3 Link</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>COVID Chest X-Ray</td>
<td>Identify COVID by chest x-ray pictures</td>
<td>599 / 75</td>
<td>2</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Bottles</td>
<td>Identify how full a soda bottle is</td>
<td>11992 / 1499</td>
<td>5</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Indoor Scenes</td>
<td>Identify the kind of indoor place</td>
<td>2498 / 312</td>
<td>10</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Flowers</td>
<td>Recognize flower types</td>
<td>3455 / 431</td>
<td>5</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Four Shapes</td>
<td>Identify basic shapes</td>
<td>11976 / 1496</td>
<td>4</td>
<td>1 Link</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Fruits</td>
<td>Identify fruits in different lighting conditions</td>
<td>35091 / 4386</td>
<td>15</td>
<td>1 Link</td>
<td></td>
</tr>
</tbody>
</table>

*Continued on next page*
<table>
<thead>
<tr>
<th>No.</th>
<th>Dataset Name</th>
<th>Brief Description</th>
<th>Instances</th>
<th>Cls.</th>
<th>Splits</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaTrain-34</td>
<td>Fruits 360</td>
<td>Identify fruits in various orientations</td>
<td>72225 / 9057</td>
<td>131</td>
<td>7</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-35</td>
<td>Garbage</td>
<td>Classify garbage types</td>
<td>2019 / 252</td>
<td>6</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-36</td>
<td>Handwritten digits</td>
<td>Identify handwritten digits</td>
<td>47995 / 5999</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-37</td>
<td>Emojis</td>
<td>Identify the type of emojis from various styles</td>
<td>5324 / 667</td>
<td>50</td>
<td>3</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-38</td>
<td>German Traffic Signs</td>
<td>Classify german traffic signs</td>
<td>31367 / 3921</td>
<td>43</td>
<td>3</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-39</td>
<td>Flowers 2</td>
<td>Identify type of flower</td>
<td>5194 / 651</td>
<td>102</td>
<td>6</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-40</td>
<td>Scraped Images</td>
<td>Classify web images into four generic categories</td>
<td>27258 / 3408</td>
<td>4</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-41</td>
<td>Natural Images</td>
<td>Classify natural images into six generic categories</td>
<td>13620 / 1703</td>
<td>6</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-42</td>
<td>Animals and Objects</td>
<td>Identify images of objects and animals</td>
<td>4000 / 500</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-43</td>
<td>Ships</td>
<td>Identify types of ships</td>
<td>3998 / 500</td>
<td>5</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-44</td>
<td>Surgical Tools</td>
<td>Classify surgical tools</td>
<td>648 / 81</td>
<td>4</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-45</td>
<td>Land Use</td>
<td>Detect kind of land use from satellite images</td>
<td>14399 / 1801</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-46</td>
<td>Lego Bricks</td>
<td>Classify Lego bricks by shape</td>
<td>5103 / 638</td>
<td>16</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-47</td>
<td>Lego Bricks 2</td>
<td>Classify Lego bricks by shape</td>
<td>7319 / 915</td>
<td>20</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-48</td>
<td>Lego Minifigures</td>
<td>Classify Lego minifigures by franchise</td>
<td>128 / 14</td>
<td>14</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-49</td>
<td>Real or Fake Legos</td>
<td>Identify off-brand lego bricks from real ones</td>
<td>36606 / 4576</td>
<td>4</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-50</td>
<td>Makeup</td>
<td>Identify whether a person is wearing makeup</td>
<td>1203 / 150</td>
<td>2</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-51</td>
<td>Male Female</td>
<td>Identify gender of a person</td>
<td>46913 / 5864</td>
<td>2</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-52</td>
<td>Messy vs Clean Room</td>
<td>Classify pictures of rooms as either messy or clean</td>
<td>168 / 22</td>
<td>2</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-53</td>
<td>Cats vs Dogs</td>
<td>Identify cats from dogs</td>
<td>19975 / 2497</td>
<td>2</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-54</td>
<td>Flowers 3</td>
<td>Identify type of flower</td>
<td>3109 / 388</td>
<td>5</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-55</td>
<td>Tiny Images</td>
<td>Identify type of object from tiny images</td>
<td>15995 / 2000</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-56</td>
<td>Mushroom classification</td>
<td>Classify mushrooms by genus</td>
<td>5312 / 662</td>
<td>9</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-57</td>
<td>Carpet Anomaly</td>
<td>Identify type of anomaly on carpets</td>
<td>315 / 41</td>
<td>6</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-58</td>
<td>Grid Anomaly</td>
<td>Identify type of anomaly on a grid</td>
<td>270 / 33</td>
<td>6</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-59</td>
<td>Leather Anomaly</td>
<td>Identify type of anomaly on leather</td>
<td>293 / 38</td>
<td>6</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-60</td>
<td>Natural Images 2</td>
<td>Identify types of natural images</td>
<td>11221 / 1402</td>
<td>6</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-61</td>
<td>Printed Letters</td>
<td>Identify printed Latin letters in various fonts</td>
<td>381052 / 47630</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-62</td>
<td>Bengali Digits</td>
<td>Identify handwritten Bengali digits</td>
<td>57620 / 7203</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-63</td>
<td>Flowers 4</td>
<td>Identify type of flower</td>
<td>2855 / 356</td>
<td>5</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-64</td>
<td>Oregon Wildlife</td>
<td>Identify type of wildlife in pictures taken in Oregon</td>
<td>5655 / 708</td>
<td>20</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-65</td>
<td>Parkinson's Drawings</td>
<td>Identify stage of Parkinson's disease by drawing</td>
<td>162 / 20</td>
<td>2</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-66</td>
<td>Dogs 2</td>
<td>Classify types of dogs</td>
<td>7216 / 902</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>No.</td>
<td>Dataset Name</td>
<td>Brief Description</td>
<td>Instances</td>
<td>Cls.</td>
<td>Splits</td>
<td>URL</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------</td>
<td>----------------</td>
<td>------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>MetaTrain-67</td>
<td>Seedlings</td>
<td>Determine type of plant from a picture of its seedling</td>
<td>3792 / 474</td>
<td>12</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-68</td>
<td>Traffic Signs</td>
<td>Identify traffic signs</td>
<td>5735 / 717</td>
<td>8</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-69</td>
<td>Real vs Fake Faces</td>
<td>Identify fake face images from real ones</td>
<td>1632 / 204</td>
<td>2</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-70</td>
<td>Casting Products</td>
<td>Identify defects in products manufactured by casting</td>
<td>6866 / 858</td>
<td>2</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-71</td>
<td>Rock Paper Scissors</td>
<td>Identify hand gestures</td>
<td>1749 / 219</td>
<td>3</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-72</td>
<td>Chest X-ray</td>
<td>Identify various information from chest x-ray images</td>
<td>4484 / 560</td>
<td>2</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-73</td>
<td>Furniture 2</td>
<td>Classify type of furniture</td>
<td>2400 / 200</td>
<td>200</td>
<td>10</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-74</td>
<td>Sheep</td>
<td>Classify sheep breeds</td>
<td>1344 / 168</td>
<td>4</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-75</td>
<td>Simpsons</td>
<td>Identify characters from a popular TV show</td>
<td>15969 / 1999</td>
<td>39</td>
<td>2</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-76</td>
<td>Simpsons 2</td>
<td>Identify characters from a popular TV show</td>
<td>16709 / 2090</td>
<td>39</td>
<td>2</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-77</td>
<td>Skin Cancer</td>
<td>Classify type of Skin Cancer</td>
<td>1785 / 224</td>
<td>9</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-78</td>
<td>Signed Digits</td>
<td>Identify sign language digits</td>
<td>1644 / 208</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-79</td>
<td>Stanford Dogs</td>
<td>Identify dog breeds</td>
<td>16376 / 2050</td>
<td>120</td>
<td>6</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-80</td>
<td>Preprocessed Stanford Dogs</td>
<td>Preprocessed version of Stanford Dogs</td>
<td>16418 / 2052</td>
<td>120</td>
<td>6</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-81</td>
<td>Synthetic Digits</td>
<td>Identify digits on randomly generated backgrounds</td>
<td>9600 / 1200</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-82</td>
<td>Ethiopic Digits</td>
<td>Classify Ethiopic Digits</td>
<td>48000 / 6000</td>
<td>10</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-83</td>
<td>Simpsons 3</td>
<td>Identify characters from a TV show</td>
<td>16709 / 2090</td>
<td>39</td>
<td>2</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-84</td>
<td>Traffic Signs 2</td>
<td>Identify traffic signs</td>
<td>20288 / 2530</td>
<td>67</td>
<td>4</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-85</td>
<td>Vehicles</td>
<td>Classify types of auto vehicles</td>
<td>22427 / 2803</td>
<td>17</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTrain-86</td>
<td>Clothes</td>
<td>Identify types of clothes</td>
<td>12935 / 1617</td>
<td>6</td>
<td>1</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-1</td>
<td>Alien vs Predator</td>
<td>Tell apart characters from a movie</td>
<td>711 / 89</td>
<td>2</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-2</td>
<td>Colorectal Histology</td>
<td>Classify colorectal tissue images</td>
<td>4000 / 496</td>
<td>8</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-3</td>
<td>COVID-19</td>
<td>Identify lung diseases from radiographic images</td>
<td>2298 / 288</td>
<td>3</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-4</td>
<td>Speed Limit Signs</td>
<td>Classify road speed limit signs</td>
<td>272 / 35</td>
<td>4</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-5</td>
<td>Gemstones</td>
<td>Classify different kinds of gemstones</td>
<td>2206 / 278</td>
<td>18 (87)†</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-6</td>
<td>Honeybee Pollen</td>
<td>Detect whether a honeybee is carrying pollen</td>
<td>571 / 71</td>
<td>2</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-7</td>
<td>Chinese Characters</td>
<td>Identify handwritten Chinese characters</td>
<td>13762 / 1715</td>
<td>20 (200)†</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-8</td>
<td>Real or Drawing</td>
<td>Identify real images from drawings in tiny images</td>
<td>4000 / 500</td>
<td>10</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-9</td>
<td>Dessert</td>
<td>Identify types of dessert</td>
<td>1324 / 166</td>
<td>5</td>
<td>-</td>
<td>Link</td>
</tr>
<tr>
<td>MetaTest-10</td>
<td>Dog Breeds</td>
<td>Classify dog breeds</td>
<td>5295 / 656</td>
<td>19 (133)†</td>
<td>-</td>
<td>Link</td>
</tr>
</tbody>
</table>

†The original dataset’s number of classes are written in parentheses.