NAS-OoD: Neural Architecture Search for Out-of-Distribution Generalization

Haoyue Bai
The Hong Kong University of Science and Technology
hbaiaa@cse.ust.hk

Fengwei Zhou
Huawei Noah's Ark Lab
zhoufengwei@huawei.com

Lanqing Hong
Huawei Noah's Ark Lab
honglanqing@huawei.com

Nanyang Ye *
Shanghai Jiao Tong University
ynylincoln@sjtu.edu.cn

S.-H. Gary Chan
The Hong Kong University of Science and Technology
gchan@cse.ust.hk

Zhenguo Li
Huawei Noah's Ark Lab
li.zhenguo@huawei.com

Abstract

Recent advances on Out-of-Distribution (OoD) generalization reveal the robustness of deep learning models against distribution shifts. However, existing works focus on OoD algorithms, such as invariant risk minimization, domain generalization, or stable learning, without considering the influence of deep model architectures on OoD generalization, which may lead to sub-optimal performance. Neural Architecture Search (NAS) methods search for architecture based on its performance on the training data, which may result in poor generalization for OoD tasks. In this work, we propose robust Neural Architecture Search for OoD generalization (NAS-OoD), which optimizes the architecture with respect to its performance on generated OoD data by gradient descent. Specifically, a data generator is learned to synthesize OoD data by maximizing losses computed by different neural architectures, while the goal for architecture search is to find the optimal architecture parameters that minimize the synthetic OoD data losses. The data generator and the neural architecture are jointly optimized in an end-to-end manner, and the minimax training process effectively discovers robust architectures that generalize well for different distribution shifts. Extensive experimental results show that NAS-OoD achieves superior performance on various OoD generalization benchmarks with deep models having a much fewer number of parameters. In addition, on a real industry dataset, the proposed NAS-OoD method reduces the error rate by more than 70% compared with the state-of-the-art method, demonstrating the proposed method's practicality for real applications.

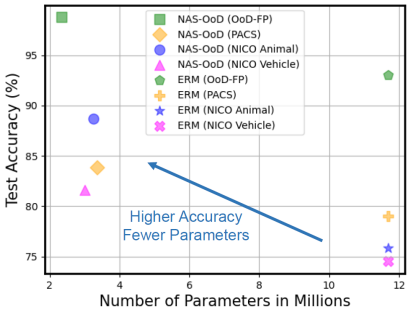


Figure 1. NAS-OoD performs significantly better than existing OoD generalization baselines in terms of test accuracy and network parameter numbers. The upper left points are better than lower right ones because they have higher test accuracy and lower parameter numbers.

1. Introduction

Deep learning models have encountered significant performance drop in Out-of-Distribution (OoD) scenarios [4, 26], where test data come from a distribution different from that of the training data. With their growing use in real-world applications in which mismatches of test and training data distributions are often observed [25], extensive efforts have been devoted to improving generalization ability [30, 3, 20, 5]. Risk regularization methods [3, 1, 41] aim to learn invariant representations across different training environments by imposing different invariant risk regularization. Domain generalization methods [30, 31, 9, 48] learn models from multiple domains such that they can generalize well to unseen domains. Stable learning [27, 28, 20] focuses on identifying stable and causal features for predictions. Existing works, however, seldom consider the effects

*Nanyang Ye is the corresponding author.

of architectures on generalization ability. On the other hand, some pioneer works suggest that different architectures show varying OoD generalization abilities [22, 12, 33]. How a network's architecture affects its ability to handle OoD distribution shifts is still an open problem.

Conventional Neural Architecture Search (NAS) methods search for architectures with maximal predictive performance on the validation data that are randomly divided from the training data [49, 36, 34, 44]. The discovered architectures are supposed to perform well on unseen test data under the assumption that data are Independent and Identically Distributed (IID). While novel architectures discovered by recent NAS methods have demonstrated superior performance on different tasks with the IID assumption [40, 18, 45, 24], they may suffer from over-fitting in OoD scenarios, where the test data come from another distribution. A proper validation set that can evaluate the performance of architectures on the test data with distribution shifts is crucial in OoD scenarios.

In this paper, we propose robust NAS for OoD generalization (NAS-OoD) that searches architectures with maximal predictive performance on OoD examples generated by a conditional generator. An overview of the proposed method is illustrated in Figure 2. To do NAS and train an OoD model simultaneously, we follow the line of gradient-based methods for NAS [34, 42, 8, 23, 44], however, we extend that on several fronts. The discrete selection of architectures is relaxed to be differentiable by building all candidate architectures into a supernet with parameter sharing and adopting a softmax choice over all possible network operations. The goal for architecture search is to find the optimal architecture parameters that minimize the validation loss under the condition that the corresponding network parameters minimize the training loss.

Instead of using part of the training set as the validation set, we train a conditional generator to map the original training data to synthetic OoD examples as the validation data. The parameters of the generator are updated to maximize the validation loss computed by the supernet. This update encourages the generator to synthesize data having a different distribution from the original training data since the supernet is optimized to minimize the error on the training data. To search for the architectures with optimal OoD generalization ability, the architecture parameters are optimized to minimize the loss on the validation set containing synthetic OoD data. This minimax training process effectively drives both the generator and architecture search to improve their performance and finally derive the robust architectures that perform well for OoD generalization.

Our main contributions can be summarized as follows:

1. To the best of our knowledge, NAS-OoD is the first attempt to introduce NAS for OoD generalization, where a conditional generator is jointly optimized to synthesize OoD examples helping to correct the supervisory signal for architecture search.

2. NAS-OoD gets the optimal architecture and all optimized parameters in a single run. The minimax training process effectively discovers robust architectures that generalize well for different distribution shifts.

3. We take the first step to understanding the OoD generalization of neural network architectures systematically. We provide a statistical analysis of the searched architectures and our preliminary practice shows that architecture does influence OoD robustness.

4. Extensive experimental results show that NAS-OoD outperforms the previous SOTA methods and achieves the best overall OoD generalization performance on various types of OoD tasks with the discovered architectures having a much fewer number of parameters.

2. Related Work

2.1. Out-of-Distribution Generalization

Data distribution mismatches between training and testing set exist in many real-world scenes. Different methods have been developed to tackle OoD shifts. IRM [3] targets to extract invariant representation from different environments via an invariant risk regularization. IRM-Games [1] aims to achieve the Nash equilibrium among multiple environments to find invariants based on ensemble methods. REx [26] proposes a min-max procedure to deal with the worst linear combination of risks across different environments. MASF [15] adopts a framework to learn invariant features among domains. JiGen [9] jointly classifies objects and solves unsupervised jigsaw tasks. CuMix [35] aims to recognize unseen categories in unseen domains through a curriculum procedure to mix up data and labels from different domains. DecAug [5] proposes a decomposed feature representation and semantic augmentation approach to address diversity and distribution shifts jointly. The work of [21] finds that using pre-training can improve model robustness and uncertainty. However, existing OoD generalization approaches seldom consider the effects of architecture which leads to suboptimal performances. In this work, we propose NAS-OoD, a robust network architecture search method for OoD generalization.

2.2. Neural Architecture Search

EfficientNet [40] proposes a new scaling method that uniformly scales all dimensions of depth, width, and resolution via an effective compound coefficient. EfficientNet design a new baseline which achieves much better accuracy and efficiency than previous ConvNets. One-shot NAS [6]

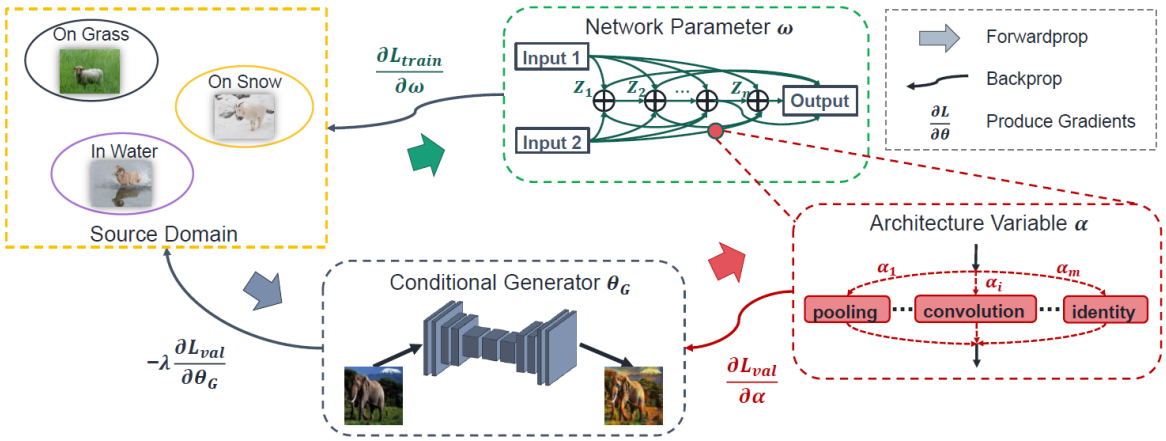


Figure 2. An overview of the proposed NAS-OoD. A conditional generator is learned to map the original training data to synthetic OoD examples by maximizing their losses computed by different neural architectures. Meanwhile, the architecture search process is optimized by minimizing the synthetic OoD data losses.

discusses the weight sharing scheme for one-shot architecture search and shows that it is possible to identify promising architectures without either hypernetworks or RL efficiently. DARTS [34] presents a differentiable manner to deal with the scalability challenge of architecture search. ISTA-NAS [44] formulates neural architecture search as a sparse coding problem. In this way, the network in search satisfies the sparsity constraint at each update and is efficient to train. SNAS [42] reformulates NAS as an optimization problem on parameters of a joint distribution for the search space in a cell. DSNAS [23] proposes an efficient NAS framework that simultaneously optimizes architecture and parameters with a low-biased Monte Carlo estimate. NASDA [33] leverages a principle framework that uses differentiable neural architecture search to derive optimal network architecture for domain adaptation tasks. NADS [2] learns a posterior distribution on the architecture search space to enable uncertainty quantification for better OoD detection and aims to spot anomalous samples. The work [10] uses a robust loss to mitigate the performance degradation under symmetric label noise. However, NAS overfits easily, the work [43, 19] points out that NAS evaluation is frustratingly hard. Thus, it is highly non-trivial to extend existing NAS algorithms to the OoD setting.

2.3. Robustness from Architecture Perspective

Recent studies show that different architectures present different generalization abilities. The work of [46] uses a functional modular probing method to analyze deep model structures under the OoD setting. The work [22] examines and shows that pre-trained transformers achieve not only high accuracy on in-distribution examples but also improvement of out-of-distribution robustness. The work [12] presents CNN models with neural hidden layers that better simulate the primary visual cortex improve robustness against image perturbations. The work [14] uses a pure transformer applied directly to sequences of image patches, which performs quite well on image classification tasks compared with relying on CNNs. The work of [13] targets to improve the adversarial robustness of the network with NAS and achieves superior performance under various attacks. However, they do not consider OoD generalization from the architecture perspective.

3. Methodology

In this section, we first introduce preliminaries on conventional NAS and their limitations in OoD scenarios (Section 3.1). Then, we describe the details of our robust Neural Architecture Search for OoD generalization (Section 3.2).

3.1. Preliminaries on Differentiable Neural Architecture Search

Conventional NAS methods mainly search for computation cells as the building units to construct a network [34, 42, 23]. The search space of a cell is defined as a directed acyclic graph with n ordered nodes $\{z_1, z_2, ..., z_n\}$ and edges $\xi = \{e^{i,j} | 1 \leq i < j \leq n\}$. Each edge includes m candidate network operations chosen from a predefined operation space $\mathcal{O} = \{o_1, o_2, ..., o_m\}$, such as max-pooling, identity and dilated convolution. The binary variable $s_k^{(i,j)} \in \{0, 1\}$ denotes the corresponding active connection. Thus, the node can be formed as:

$$z_j = \sum_{i=1}^{j-1} \sum_{k=1}^{m} s_k^{(i,j)} o_k(z_i) = \boldsymbol{s}_j^T \boldsymbol{o}_j, \tag{1}$$

where $\boldsymbol{s}_j$ is the vector consists of $s_k^{(i,j)}$ and $\boldsymbol{o}_j$ denotes the vector formed by $o_k(z_i)$. As the binary architecture variables s_j is hard to optimize in a differentiable way, recent

DARTS-based NAS methods make use of the continuous relaxation in the form of

$$s_k^{(i,j)} = \exp(\alpha_k^{(i,j)}) / \sum_k \exp(\alpha_k^{(i,j)}), \tag{2}$$

and optimize $\alpha_k^{(i,j)}$ as trainable architecture parameters [34], which can be formulated as the following bilevel optimization problem:

$$\begin{aligned} &\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \ \ell_{\text{val}}(\boldsymbol{\omega}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}), \\ &s.t. \quad \boldsymbol{\omega}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{\omega}} \ \ell_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha}), \end{aligned} \tag{3}$$

where $\boldsymbol{\alpha}$ denotes the architecture variable vector formed by $\alpha_k^{(i,j)}$, and $\boldsymbol{\omega}$ denotes the supernet parameters. ℓ_{train} and ℓ_{val} denote the training and validation losses, respectively. In the search phase, $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ are optimized in an alternate manner.

The validation data used for the above architecture search method are usually divided from the training data. Previous research demonstrates that the derived architectures perform well on different tasks [34, 42, 23] when the training and test data are IID. However, when dealing with OoD tasks, where the test data come from another distribution, using part of the training set as the validation set may cause NAS methods suffer from over-fitting and the searched architectures to be sub-optimal in this situation. Thus, a proper validation set is needed to effectively evaluate the performance of discovered architectures on the test set in OoD scenarios.

3.2. NAS-OoD: Neural Architecture Search for OoD Generalization

In OoD learning tasks, we are provided with K source domains. The target is to discover the optimal network architecture that can generalize well to the unseen target domain. In the following descriptions, let $\boldsymbol{\alpha}$, $\boldsymbol{\omega}$ and $\boldsymbol{\theta}_G$ denote the parameters for architecture topology, the supernet and the conditional generator $G(\cdot,\cdot)$, respectively. The conditional generator $G(\cdot,\cdot)$ takes data $\boldsymbol{x}$ and domain labels $\tilde{\boldsymbol{k}}$ as the input. Let ℓ_{train} be the training loss function, and ℓ_{val} be the validation loss function.

Minimax optimization for NAS-OoD. To generate a proper validation set for OoD generalization in NAS, as shown in Figure 2, a conditional generator is learned to generate novel domain data by maximizing the losses on different neural architectures, while the optimal architecture variables are optimized by minimizing the losses on generated OoD images. This can be formulated as a constrained minimax optimization problem as follows:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \max_{G} \ell_{\text{val}}(\boldsymbol{\omega}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}, G(\boldsymbol{x}, \tilde{\boldsymbol{k}})), \\ &\text{s.t.} \quad \boldsymbol{\omega}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{\omega}} \ell_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha}, \boldsymbol{x}), \end{aligned} \tag{4}$$

Algorithm 1 NAS-OoD: Neural Architecture Search for OoD generalization

Input: Training set $\mathcal{D}$, batch size n, learning rate μ.
Output: $\boldsymbol{\alpha}, \boldsymbol{\omega}, \boldsymbol{\theta}_G$.

Initialize $\boldsymbol{\alpha}, \boldsymbol{\omega}, \boldsymbol{\theta}_G$;
repeat
  Sample a mini-batch of training images $\{(x_i, y_i)\}_{i=1}^n$;
  Generate novel domain data: $x_i^{\text{syn}} \leftarrow \text{G}(x_i, \tilde{\boldsymbol{k}})$;
  $\boldsymbol{\theta}_G \leftarrow \boldsymbol{\theta}_G - \mu \cdot \nabla_{\boldsymbol{\theta}_G} \ell_{\text{aux}}$ according to Eqn. (8);
  $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \mu \cdot \nabla_{\boldsymbol{\omega}} \ell_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha}, x_i)$ according to Eqn. (5);
  $\boldsymbol{\theta}_G \leftarrow \boldsymbol{\theta}_G + \mu \cdot \nabla_{\boldsymbol{\theta}_G} \ell_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha}, x_i^{\text{syn}})$ according to Eqn. (5);
  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \mu \cdot \nabla_{\boldsymbol{\alpha}} \ell_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha}, x_i^{\text{syn}})$ according to Eqn. (5);
until convergence;

where $G(\boldsymbol{x}, \tilde{\boldsymbol{k}})$ is the generated data from the original input data $\boldsymbol{x}$ on the novel domain $\tilde{\boldsymbol{k}}$. This is different from NAS methods' formulation as we introduce a generator to adversarially generate challenging data from original input for validation loss to search for network architectures. This can avoid over-fitting problem by using the same data for optimizing neural network parameters and architectures as shown in our experiment. Solving this problem directly will involve calculating second-order derivatives that will bring much computational overhead and the constraint is hard to realize. Thus, we introduce the following practical implementations of our algorithm.

Practical implementation. Inspired by the previous work in meta-learning [16], we approximate the multi-step optimization with the one-step gradient when calculating the gradient for $\boldsymbol{\alpha}$. Different source domains are mixed together in architecture search, while the domain labels are embedded in the generator auxiliary loss training process which will be explained later. For the architecture search training process, architecture parameters $\boldsymbol{\alpha}$, network parameters $\boldsymbol{\omega}$ and parameters for conditional generator $\boldsymbol{\theta}_G$ are updated in an iterative training process:

$$\begin{aligned} &\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \mu \cdot \nabla_{\boldsymbol{\omega}} \ell_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha}, \boldsymbol{x}), \\ &\boldsymbol{\theta}_G \leftarrow \boldsymbol{\theta}_G + \mu \cdot \nabla_{\boldsymbol{\theta}_G} \ell_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha}, G(\boldsymbol{x}, \tilde{\boldsymbol{k}})), \\ &\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \mu \cdot \nabla_{\boldsymbol{\alpha}} \ell_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha}, G(\boldsymbol{x}, \tilde{\boldsymbol{k}})). \end{aligned} \tag{5}$$

Generator's auxiliary losses. To train the generator and improve consistency, we apply an additional cycle consistency constraint to the generator:

$$\ell_{\text{cycle}} = ||\text{G}(\text{G}(\boldsymbol{x}_k, \widetilde{k}), k) - \boldsymbol{x}_k||_1, \tag{6}$$

where $\boldsymbol{x}_k$ denotes data from K source domains with domain $\{s_1, s_2, ..., s_K\}$, $\widetilde{k}$ denotes the domain index for the generated novel domain s_{K+1}, and $||\cdot||_1$ refers to L1 norm. This can regularize the generator to be able to produce data from and back to the source domains.

To preserve semantic information, we also require the generated data $\mathrm{G}(\boldsymbol{x}_k, \widetilde{k})$ to keep the same category as the original data $\boldsymbol{x}_k$.

$$\ell_{\text{ce}} = \text{CE}(Y(\mathrm{G}(\boldsymbol{x}_k, \widetilde{k})), Y^*(\boldsymbol{x}_k)), \tag{7}$$

where CE be the cross-entropy loss, Y is a classifier with a few convolutional layers pretrained on training data, $Y^*(\cdot)$ is the ground-truth labels for the input data.

The total auxiliary loss for generator is defined as follows:

$$\ell_{\text{aux}} = \lambda_{cycle} \cdot \ell_{\text{cycle}} + \lambda_{ce} \cdot \ell_{\text{ce}}, \tag{8}$$

where λ_{ce} and λ_{cycle} are hyper-parameters.

Compared with the gradient-based perturbation [38], the conditional generator is able to model a more sophisticated distribution shift due to its intrinsic learnable nature. The NAS-OoD algorithm is outlined in Algorithm 1.

4. Experiments

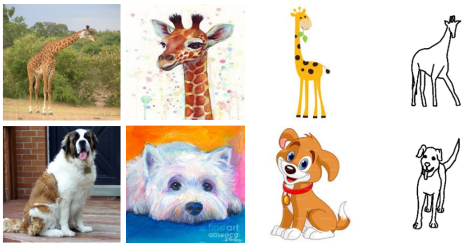
In this section, we conduct numerical experiments to evaluate the effectiveness of our proposed NAS-OoD method. To provide a comprehensive comparison with baselines, We compare our proposed NAS-OoD with the SOTA algorithms from various OoD areas, including empirical risk minimization (ERM [3]), invariant risk minimization (IRM [3]), risk extrapolation (REx [26]), domain generalization by solving jigsaw puzzles (JiGen [9]), mixup (Mixup [47]), curriculum mixup (Cumix [35]), marginal transfer learning (MTL [7]), domain adversarial training (DANN [17]), correlation alignment (CORAL [39]), maximum mean discrepancy (MMD [32]), distributionally robust neural network (DRO [37]), convnets with batch balancing (CNBB [20]), cross-gradient training (CrossGrad [38]), and the recently proposed decomposed feature representation and semantic augmentation (DecAug [5]).

For ablation studies, We also compare NAS-OoD with SOTA NAS methods, such as differentiable architecture search (DARTS [34]), stochastic neural architecture search (SNAS [42]), efficient and consistent neural architecture search by sparse coding (ISTA-NAS [44]). This is to test whether naively combining NAS methods with OoD learning algorithms can improve the generalization performance.

Our framework was implemented with PyTorch 1.4.0 and CUDA v9.0. We conducted experiments on NVIDIA Tesla V100. Following the design of [11], our generator model has an encoder-decoder structure, which consists of two down-sampling convolution layers with stride 2, three residual blocks, and two transposed convolution layers with stride 2 for up-sampling. The domain indicator is encoded as a one-hot vector. The one-hot vector is first spatially expanded and then concatenated with the input image to train the generator.

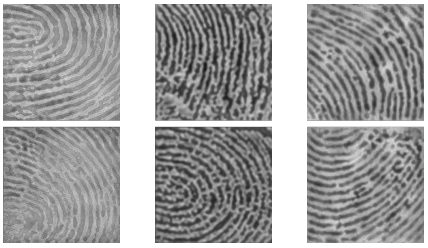
(a) In water (b) On snow (c) On grass (d) Others

Figure 3. Examples of the out-of-distribution data from the NICO dataset with contexts (a) In water, (b) On snow, (c) On grass, and(d) Others.



(a) P (b) A (c) C (d) S

Figure 4. Typical examples of out-of-distribution data with diversity shift from the PACS dataset. (a) Photo. (b) Art Painting. (c) Cartoon. (d) Sketch.



(a) Domain 1 (b) Domain 2 (c) Domain 3

Figure 5. Typical examples of out-of-distribution data in the OoD-FP dataset with three different domains.

4.1. Evaluation Datasets

We evaluate our NAS-OoD on four challenging OoD datasets: NICO Animal, NICO Vehicle, PACS, and Out-of-Distribution Fingerprint (OoD-FP) dataset where methods have to be able to perform well on data distributions different from training data distributions. The evaluation metric is the classification accuracy of the test set. The number of neural network parameters is used to measure the computational complexity for comparison between different neural network architectures.

NICO (Non-I.I.D. Image with Contexts) dataset: NICO consists of two datasets, i.e., NICO Animal with 10 classes and NICO Vehicle with 9 classes. The NICO dataset is a recently proposed OoD generalization dataset in the real sce-

Table 1. Results of NAS-OoD compared with different methods with ResNet-18 (11.7M) on the NICO dataset.

Model	Animal	Vehicle	Average
ERM [3]	75.87	74.52	75.19
IRM [3]	59.17	62.00	60.58
REx [26]	74.31	66.20	70.25
JiGen [9]	84.95	79.45	82.20
Mixup [47]*	80.27	77.00	78.63
Cumix [35]	76.78	74.74	75.76
MTL [7]*	78.89	75.11	77.00
DANN [17]	75.59	72.23	73.91
CORAL [39]*	80.27	71.64	75.95
MMD [32]*	70.91	68.04	69.47
DRO [37]*	77.61	74.59	76.10
CNBB [20]	78.16	77.39	77.77
DecAug [5]	85.23	80.12	82.67
NAS-OoD	***88.72***	***81.59***	***85.16***
Params (M)	3.25	3.00	3.13

* Implemented by ourselves.

narios [20] (see Figure 3), which contains different contexts, such as different object poses, positions, and backgrounds across the training, validation, and test sets.

PACS (Photo, Art painting, Cartoon, Sketch) dataset: This dataset is commonly used in OoD generalization (see Figure 4). It contains four domains with different image styles, namely photo, art painting, cartoon, and sketch with seven categories (dog, elephant, giraffe, guitar, horse, house, person). We follow the same leave-one-domain-out validation experimental protocol as in [30], i.e., we select three domains for training and the remaining domain for testing for each time.

OoD-FP (Out-of-Distribution Fingerprint) dataset: The OoD-FP dataset is a real industry dataset that contains three domains corresponding to different fingerprint collection devices on different brands of mobile phones (see Figure 5). In the fingerprint recognition task, the goal is to learn to distinguish whether input fingerprints are from the users' fingerprints stored in the dataset. Due to the hardware implementation differences, fingerprints exhibit different styles from different devices. In our setting, the goal is to learn a universal fingerprint recognition neural network to generalize on the fingerprints collected from unseen datasets.

4.2. Results and Discussion

NAS-OoD achieves the SOTA performance *simultaneously* on various datasets from different OoD research areas, such as stable learning, domain generalization, and real industry dataset.

The results for the challenging NICO dataset are shown in Table 1. From Table 1, the proposed NAS-OoD method achieves the SOTA performance simultaneously on the two subsets of the NICO dataset with a much fewer number of parameters. Specifically, NAS-OoD achieves 88.72% on NICO Animal and 81.59% on NICO Vehicle with only around 3.1 million parameters compared with DecAug achieving 82.67% accuracy but with 11.7 million parameters. The superior performance of NAS-OoD also confirms the possibility of improving the neural network's OoD generalization performance by searching for the architecture, which provides an orthogonal way to improve the OoD generalization.

Table 2. Classification accuracy on the PACS dataset compared with different methods with ResNet-18 (11.7M).

Model	A	C	S	P	Average
ERM [3]	77.85	74.86	67.74	95.73	79.05
IRM [3]	70.31	73.12	75.51	84.73	75.92
REx [26]	76.22	73.76	66.00	95.21	77.80
JiGen [9]	79.42	75.25	71.35	96.03	80.51
Mixup [47]*	82.01	72.58	72.48	93.29	80.09
CuMix [35]	82.30	76.50	72.60	95.10	81.60
MTL [7]*	76.76	71.87	76.73	92.65	79.50
MLDG [29]	79.50	77.30	71.50	94.30	80.70
MASF [15]	80.29	77.17	71.69	94.99	81.03
DANN [17]	81.30	73.80	74.30	94.00	80.80
CORAL [39]*	80.49	74.32	75.06	94.09	80.99
MMD [32]*	79.34	73.76	72.61	94.19	79.97
DRO [37]*	78.09	74.18	77.00	93.45	80.68
CrossGrad [38]	78.70	73.30	65.10	94.00	80.70
L2A-OT [48]	83.30	78.20	73.60	96.20	82.80
DecAug [5]	79.00	79.61	75.64	95.33	82.39
NAS-OoD	***83.74***	***79.69***	***77.27***	***96.23***	***84.23***
Params (M)	3.51	3.44	3.35	3.15	3.36

* Implemented by ourselves.

We also compare our methods with different domain generalization methods on the PACS dataset. The results are shown in Table 2. Similarly, we observe that NAS-OoD achieves SOTA performance on all the four domains and the best average generalization performance of 83.89% with only 3.36 million of network parameters. The generalization accuracy is much better than previous OoD algorithms DecAug (82.39%), JiGen (80.51%), IRM (75.92%) with ResNet-18 backbone, which are the best OoD approaches before NAS-OoD. The network parameters for ResNet-18 is 11.7 million which is much larger than the network searched by our NAS-OoD. Note that the relative performance for some algorithms may change drastically between NICO and PACS datasets whereas the proposed NAS-OoD algorithm can generalize well simultaneously on datasets from different OoD research areas.

To test the generalization performance of NAS-OoD on real industry datasets, we compare NAS-OoD with other methods on OoD-FP dataset. The results are shown in Ta-

Table 3. Classification accuracy compared to different methods with ResNet-18 backbone (11.7M) on the OoD-FP dataset. All methods are implemented by ourselves.

Model	Domain 1	Domain 2	Domain 3	Average
ERM [3]	93.75	92.70	92.70	93.05
IRM [3]	95.83	87.50	84.37	89.23
REx [26]	97.91	91.66	92.70	94.09
Mixup [47]	96.87	97.91	90.62	95.13
MTL [7]	95.83	97.91	90.62	94.78
DANN [17]	95.83	97.91	86.45	93.39
CORAL [39]	94.79	97.91	91.66	94.78
MMD [32]	96.87	95.83	94.79	95.83
DRO [37]	96.87	95.83	89.58	94.09
NAS-OoD	***99.27***	***99.49***	***97.54***	***98.77***
Params (M)	2.28	2.28	2.43	2.33

ble 3. NAS-OoD consistently achieves good generalization performance with the non-trivial improvement compared with other methods. NAS-OoD achieves a 1.23% error rate in the fingerprint classification task which almost reduces the error rate by around 70% compared with the second-best method–MMD. This demonstrates the superiority of NAS-OoD and especially its potential to be practically useful in real industrial applications.

4.3. Ablation Study

In this section, we first test whether naively combining NAS methods with domain generalization methods can achieve good OoD generalization performances. We conduct experiments on the NICO dataset. The results are shown in Table 4. It can be seen that using NAS methods only, such as DARTS, can achieve only 79.61% average accuracy, significantly lower than most compared compositions. This is in stark contrast with the good performance for NAS methods on IID generalization tasks where training and test datasets have similar distributions. This is because NAS methods are doing variational optimization by finding not only the best parameters but also the best function for fitting whereas this can help NAS methods to achieve good performance in IID settings. In OoD settings, where test data distributions differ significantly from training data distributions, NAS methods can overfit the training data distribution and achieve sub-optimal performance. Besides, we can also observe that naively combining the NAS methods with OoD learning algorithms, such as IRM, brings no statistically significant performance gain. The reason is that many OoD learning algorithms are based on implicit or explicit regularization added to the ERM loss. NAS methods will explore the search space to fit the loss terms and the regularization term may be ignored as NAS methods may exploit only the ERM loss, thus bringing no performance gain. This also confirms that generating OoD data is needed during training to avoid over-fitting.

Table 4. Results of NAS-OoD compared with other NAS methods. The baselines are implemented by ourselves.

Model	Animal	Vehicle	Average
DARTS [34]	83.67	75.55	79.61
DARTS + IRM [34]	82.29	72.24	77.26
SNAS [42]	85.96	80.04	83.00
SNAS + IRM [42]	82.94	76.51	79.73
ISTA-NAS [44]	86.70	80.56	83.63
ISTA-NAS + IRM [44]	82.57	77.61	80.09
NAS-OoD	***88.72***	***81.59***	***85.16***

Table 5. NAS-OoD variants.

Model	Animal	Vehicle	Average
Random Sample	80.92	76.43	78.68
NAS-OoD w/o cycle loss	86.88	80.85	83.86
NAS-OoD	***88.72***	***81.59***	***85.16***

As shown in Table 5, the average results of randomly sampled architectures are 79.45% (Animal) and 75.70% (Vehicle), which is significantly lower than those of the architectures searched by NAS-OoD. We also conduct an ablation study on this auxiliary loss, the average accuracy on NICO without cycle loss is 83.86%, which is low than our proposed method 85.16%. This shows the effectiveness of the auxiliary cycle loss to facilitate the searching process.

4.4. Analysis of Searched Architectures

After setting up the NAS-OoD framework, we want to analyze whether any special patterns for searched cell-based network architectures and whether the NAS-OoD framework can stably find consistent architectures.

Temporal stability of searched architecture To check whether the found special pattern is consistent during the training process, we plot the operation type's percentage during the training process in Figure 6. In Figure 6, we can find that as the training proceeds, the architecture found by NAS-OoD is converging to the pattern that the percentage of dilated convolution 3×3 is higher and the separable convolution 3×3 is lower. This might be because dilated convolution has a larger receptive field than separable convolution which only receives one channel for each convolution kernel and a large receptive field can better learn the shape features of objects rather than the spurious features, such as color and texture.

Cross dataset architecture patterns To check whether the architecture patterns searched by NAS-OoD are similar across different datasets, we plot the operation type's percentage for different datasets in Figure 7.

We found there are similarities of architectures found on different datasets. Specifically, the searched NAS-OoD ar-

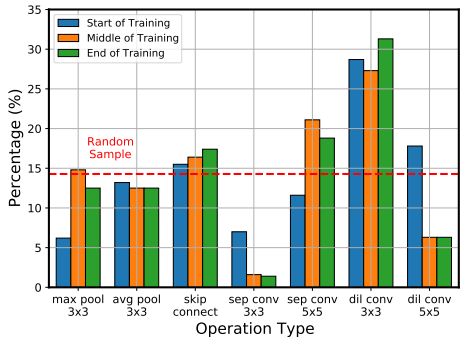


Figure 6. Temporal stability of search architecture.(Better viewed in the zoom-in mode)

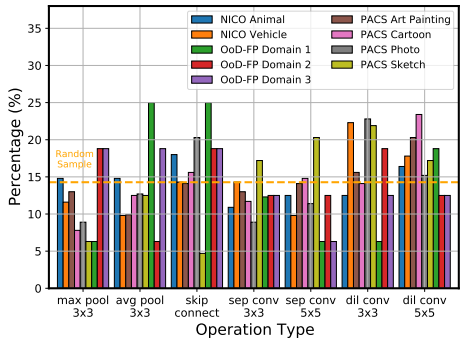
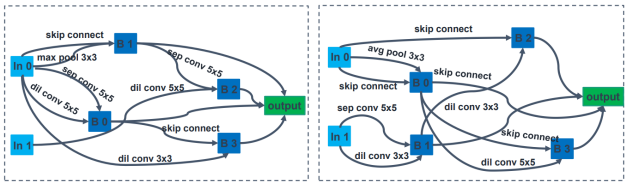


Figure 7. Statistical analysis of searched architectures on different datasets.(Better viewed in the zoom-in mode)

chitectures tend to contain more convolutional operations with a large kernel size compared with randomly sampled architectures. This may be because a larger kernel size convolution operation has larger receptive fields, which makes better use of contextual information compared with a small kernel size. NAS-OoD architectures also present more skip connection operations compared with random selection and locate on both skip edges and direct edges, which can better leverage both low-level texture features and high-level semantic information for recognition. There is some previous study show that densely connected pattern benefits model robustness. NAS-OoD searched for more dilated convolutions than normal convolutions, which may be due to that the dilated convolutions enlarge the receptive fields.

Visualization of the searched architecture cells. As illustrated in Figure 8, we present the detailed structures of the best cells discovered on different datasets using NAS-OoD. Figure 8 (a) show the normal cells and (b) demonstrate the reduce cells. The searched cell contains two input nodes, four intermediate nodes, and one output node. Each intermediate node has two edges to the previous nodes which consist of both direct edge and skip edge, and the operation is presented on each edge. Besides, all the intermediate nodes are connected and aggregated to the output node.



(a) Normal cells (b) Reduce cells

Figure 8. Typical examples of searched robust architectures on NICO dataset.(Better viewed in the zoom-in mode)

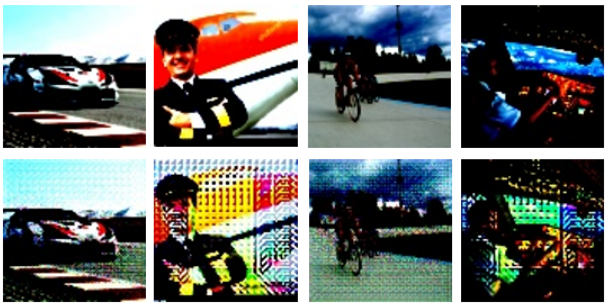


Figure 9. Some examples of synthetic images. The first row shows the original images, and the second row is its corresponding synthetic images.

Visualization of generated OoD data. We also visualize the generated OoD data in Figure 9. We find that the generated images show different properties and are clearly different from the source images. The conditional generator tends to generate images with different background patterns, textures, and colors. The semantic different make them helpful for improving out-of-distribution generalization ability.

5. Conclusions

We propose a robust neural architecture search framework that is based on differentiable NAS to understand the importance of network architecture against Out-of-Distribution robustness. We jointly optimize NAS and a conditional generator in an end-to-end manner. The generator is learned to synthesize OoD instances by maximizing their losses computed by different neural architectures, while the goal for architecture search is to find the optimal architecture parameters that minimize the synthesized OoD data losses. Our study presents several valuable observations on designing robust network architectures for OoD generalization. Extensive experiments show the effectiveness of NAS-OoD, achieving state-of-the-art performance on different OoD datasets with discovered architectures having a much fewer number of parameters.

Acknowledgements

This work was supported, in part, by Hong Kong General Research Fund (under grant number 16200120)

References

[1] Kartik Ahuja, Karthikeyan Shanmugam, Kush Varshney, and Amit Dhurandhar. Invariant risk minimization games. *arXiv:2002.04692*, 2020. 1, 2

[2] Randy Ardywibowo, Shahin Boluki, Xinyu Gong, Zhangyang Wang, and Xiaoning Qian. Nads: Neural architecture distribution search for uncertainty awareness. In *ICML*, 2020. 3

[3] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv:1907.02893*, 2019. 1, 2, 5, 6, 7

[4] Hyojin Bahng, Sanghyuk Chun, Sangdoo Yun, Jaegul Choo, and Seong Joon Oh. Learning de-biased representations with biased representations. In *ICML*, 2020. 1

[5] Haoyue Bai, Rui Sun, Lanqing Hong, Fengwei Zhou, Nanyang Ye, Han-Jia Ye, S-H Gary Chan, and Zhenguo Li. Decaug: Out-of-distribution generalization via decomposed feature representation and semantic augmentation. *arXiv:2012.09382*, 2020. 1, 2, 5, 6

[6] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018. 2

[7] Gilles Blanchard, Aniket Anand Deshmukh, Urun Dogan, Gyemin Lee, and Clayton Scott. Domain generalization by marginal transfer learning. *arXiv:1711.07910*, 2017. 5, 6, 7

[8] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv:1812.00332*, 2018. 2

[9] Fabio Maria Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. Domain generalization by solving jigsaw puzzles. In *CVPR*, 2019. 1, 2, 5, 6

[10] Yi-Wei Chen, Qingquan Song, Xi Liu, PS Sastry, and Xia Hu. On robustness of neural architecture search under label noise. *Frontiers in Big Data*, 2020. 3

[11] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *CVPR*, 2018. 5

[12] Joel Dapello, Tiago Marques, Martin Schrimpf, Franziska Geiger, David D Cox, and James J DiCarlo. Simulating a primary visual cortex at the front of cnns improves robustness to image perturbations. *BioRxiv*, 2020. 2, 3

[13] Minjing Dong, Yanxi Li, Yunhe Wang, and Chang Xu. Adversarially robust neural architectures. *arXiv:2009.00902*, 2020. 3

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv:2010.11929*, 2020. 3

[15] Qi Dou, Daniel C. Castro, Konstantinos Kamnitsas, and Ben Glocker. Domain generalization via model-agnostic learning of semantic features. In *NeurIPS*, 2019. 2, 6

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. 4

[17] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, 2016. 5, 6, 7

[18] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019. 2

[19] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020. 3

[20] Yue He, Zheyan Shen, and Peng Cui. Towards non-iid image classification: A dataset and baselines. *PR*, 2020. 1, 5, 6

[21] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *ICML*, 2019. 2

[22] Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. Pre-trained transformers improve out-of-distribution robustness. *arXiv:2004.06100*, 2020. 2, 3

[23] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. Dsnas: Direct neural architecture search without parameter retraining. In *CVPR*, 2020. 2, 3, 4

[24] Yutao Hu, Xiaolong Jiang, Xuhui Liu, Baochang Zhang, Jungong Han, Xianbin Cao, and David Doermann. Nascount: Counting-by-density with neural architecture search. *arXiv:2003.00217*, 2020. 2

[25] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Sara Beery, et al. Wilds: A benchmark of in-the-wild distribution shifts. *arXiv:2012.07421*, 2020. 1

[26] David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex). *arXiv:2003.00688*, 2020. 1, 2, 5, 6, 7

[27] Kun Kuang, Peng Cui, Susan Athey, Ruoxuan Xiong, and Bo Li. Stable prediction across unknown environments. In *SIGKDD*, 2018. 1

[28] Kun Kuang, Ruoxuan Xiong, Peng Cui, Susan Athey, and Bo Li. Stable prediction with model misspecification and agnostic distribution shift. In *AAAI*, 2020. 1

[29] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. Learning to generalize: Meta-learning for domain generalization. In *AAAI*, 2018. 6

[30] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *ICCV*, 2017. 1, 6

[31] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Learning to generalize: Meta-learning for domain generalization. *arXiv:1710.03463*, 2017. 1

[32] Haoliang Li, Sinno Jialin Pan, Shiqi Wang, and Alex C Kot. Domain generalization with adversarial feature learning. In *CVPR*, 2018. 5, 6, 7

[33] Yichen Li and Xingchao Peng. Network architecture search for domain adaptation. *arXiv:2008.05706*, 2020. 2, 3

[34] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv:1806.09055*, 2018. 2, 3, 4, 5, 7

[35] Massimiliano Mancini, Zeynep Akata, Elisa Ricci, and Barbara Caputo. Towards recognizing unseen categories in unseen domains. In *ECCV*, 2020. 2, 5, 6

[36] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 2

[37] Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv:1911.08731*, 2019. 5, 6, 7

[38] Shiv Shankar, Vihari Piratla, Soumen Chakrabarti, Siddhartha Chaudhuri, Preethi Jyothi, and Sunita Sarawagi. Generalizing across domains via cross-gradient training. *arXiv:1804.10745*, 2018. 5, 6

[39] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *ECCV*, 2016. 5, 6, 7

[40] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 2

[41] Chuanlong Xie, Fei Chen, Yue Liu, and Zhenguo Li. Risk variance penalization: From distributional robustness to causality. *arXiv:2006.07544*, 2020. 1

[42] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv:1812.09926*, 2018. 2, 3, 4, 5, 7

[43] Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. *arXiv:1912.12522*, 2019. 3

[44] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *arXiv:2010.06176*, 2020. 2, 3, 5, 7

[45] Lewei Yao, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. SM-NAS: Structural-to-modular neural architecture search for object detection. *arXiv:1911.09929*, 2019. 2

[46] Dinghuai Zhang, Kartik Ahuja, Yilun Xu, Yisen Wang, and Aaron Courville. Can subnetwork structure be the key to out-of-distribution generalization? *arXiv:2106.02890*, 2021. 3

[47] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv:1710.09412*, 2017. 5, 6, 7

[48] Kaiyang Zhou, Yongxin Yang, Timothy Hospedales, and Tao Xiang. Learning to generate novel domains for domain generalization. *arXiv:2007.03304*, 2020. 1, 6

[49] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2